# Process modelling for information system description

## Thesis

# PROCESS MODELLING FOR INFORMATION SYSTEM DESCRIPTION

STEFAN K. STANCZYK

Computing Discipline, Faculty of Mathematics
The Open University

Thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Date of submission: April 1987
Date of award: 15 July 1987

Milton Keynes, April 1987

*TO MY PARENTS*

*I am greatly indebted to my supervisor, Dr Richard Maddison, for his continous encouragement and support, and for the constructive criticism he has provided over the last three years. I take it as a rare privilege to have been directed by a scientist who has shared so much of his time and his knowledge.*

*I should like to express my sincere thanks to The Open University for the research grant. I thank the Dean and the Members of the Mathematics Faculty for the excellent working environment and stimulating atmosphere. In particular, the seminars of the Information Systems Research Group and other, less formal meetings provided an excellent form of learning and broadening my experience. Mike Newton and Hugh Robinson were always ready to provide guidance and to give of their time to numerous discussions.*

*My deepest thanks to Urszula and Piotr who have always been understanding and encouraging despite my rather late working hours.*

## ABSTRACT

My previous experiences and some preliminary studies of the relevant technical literature allowed me to identify several reasons for which the current state of the database theory seemed unsatisfactory and required further research. These reasons included: insufficient formalism of data semantics, misinterpretation of NULL values, inconsistencies in the concept of the universal relation, certain ambiguities in domain definition, and inadequate representation of facts and constraints.

The commonly accepted 'sequentiality' principle in most of the current system design methodologies imposes strong restrictions on the processes that a target system is composed of. They must be algorithmic and must not be interrupted during execution; neither may they have any parallel subprocesses as their own components. This principle can no longer be considered acceptable. In very many existing systems multiple processors perform many concurrent actions that can interact with each other.

The overconcentration on data models is another disadvantage of the majority of system design methods. Many techniques pay little (or no) attention to process definition. They assume that the model of the Real World consists only of data elements and relationships among them. However, the way the processes are related to each other (in terms of precedence relation) may have considerable impact on the data model.

It has been assumed that the Real World is discretisable, i.e. it may be modelled by a structure of objects. The word *object* is to be interpreted in a wide sense so it can mean anything within the boundaries of this part of the Real World that is to be represented in the target system. An object may then denote a fact or a physical or abstract entity, or relationships between any of these, or relationships between relationships, or even a still more complex structure.

The fundamental hypothesis was formulated stating the necessity of considering the three aspects of modelling - syntax, semantics and behaviour, and these to be considered integrally.

A syntactic representation of an object within a target system is called a construct. A construct which cannot be decomposed further (either syntactically or semantically) is defined to be an atom. Any construct is a result of the following production rules: construct ::= atom | function construct; function ::= atom | construct. This syntax forms a sentential notation.

The sentential notation allows for extensive use of denotational semantics. The meaning of a construct may be defined as a function mapping from a set of syntactic constructs to the appropriate semantic domains; these in turn appear to be sets of functions since a construct may have a meaning in more than one class of objects. Because of its functional form the meaning of a construct may be derived from the meaning of its components.

The issue of system behaviour needed further investigation and a revision of the conventional model of computing. The sequentiality principle has been rejected, concurrency being regarded as a natural property of processes. A postulate has been formulated that any potential parallelism should be constructively used for data/process design and that the process structure would affect the data model. An important distinction has been made between a process declaration - considered as a form of data or an abstraction of knowledge - and a process application that corresponds to a physical action performed by a processor, according to a specific process declaration. In principle, a process may be applied to any construct - including its own representation - and it is a matter of semantics to state whether or not it is sensible to do so. The process application mechanism has been explained in terms of formal systems theory by introducing an abstract machine with two input and two output types of channels.

The system behaviour has been described by defining a process calculus. It is based on logical and functional properties of a discrete time model and provides a means to handle expressions composed of process-variables connected by logical functors. Basic terms of the calculus are: constructs and operations (equivalence, approximation, precedence, incidence, free-parallelism, strict-parallelism). Certain properties of these operations (e.g. associativity or transitivity) allow for handling large expressions. Rules for decomposing/integrating process applications, analogous in some sense to those forming the basis for structured programming, have been derived.

CONTENTS

# 1 INTRODUCTION

## 1.1 Motivation and overview

The core activity in information system development is undoubtedly modelling the relevant part of the 'Real World'. It is probably the most creative and interesting of all the stages in the development life-cycle. The analyst has to produce an image that encapsulates the relevant knowledge about the real world in a form that can be understood in several environments - human and computer. The work should include appropriate business strategy and planning: the information system objectives fitting those of the organisation.

Many factors constrain such work. People must specialise and form teams. Top managers cannot afford the time to fully think out and effectively communicate strategy, plans and objectives for every system in detail, never mind doing the relevant modelling. Human, technical and economic factors constrain and conflict. Environmental pressures, inadequate techniques, tools and training, and limited human comprehension both complicate and challenge. Moreover, the very creation of the model, which becomes a new part of the real world, makes itself - as a model of a previous state - partially obsolete.

In modelling, people aim to produce a suitable, correct, complete and consistent representation of a 'Real World' - a designated part of the real world. They need this representation in a manageable notation. The notation should fulfil many criteria: communicating clearly, seeming simple, relevant and useful; yet precise, processable and printable.

One can hardly express the entire complexity of the Real World even in a natural language. But observations and intellectual speculations have to be so expressed. *Human thinking is a derivative of human language* [94].

> 'Language is called the garment of thought; however, it should rather be, language is the flesh-garment, the body of thought' *T. Carlyle, Sartar Resartus*

Natural language, however, fails the criteria. While rich and contextual - suiting the Real World - it allows ambiguity and inaccuracy. It cannot be subjected to rigid rules of mechanistic logic. So the modelling language must itself have rules about representations, resulting in restrictions being imposed on the models created.

The process of transformation from the informal thoughts, words and actions of the Real World to a formal form with restricted representation may prevent accuracy and completeness.

Other restrictions are introduced deliberately by designers. Being interested only in certain aspects of the Real World in their models, they disregard aspects that seem unnecessary for the target system objectives. Furthermore, human factors and business strategy generalisations for future flexibility cannot normally be adequately represented.

The most general technique commonly used for modelling may be considered as some kind of discretization. In this approach a continuous Real World is replaced

by a number of distinguishable objects. Lost properties of the continuum are modelled by associations (e.g. relationships) between the objects. This otherwise sensible engineering approach has led to inappropriate apportionment of resources. Some types of objects, especially data types, were analysed in considerable depth; methods of data analysis have been extensively developed.

People believed that the distinct information types and their relationships were so inherent in the nature of the organisation and its business that data models would outlive not just day-to-day and yearly operations but also changes of strategy.

But other types of object, especially process types, received less study. While attempts to develop suitable methods for process analysis and design on the basis of data analysis and design have succeeded, the alternative approach has mainly been disregarded or unsuccessful. Although process modelling, analysis and design by structured decomposition is well understood, the technique has limited applicability - i.e. to discrete, algorithmic, non-interrupted, single-processor situations programmable using sequencing, selections and repetitions. In other situations analysts and designers flounder unskilful, haphazardly hoping for ingenuity.

Three types of objects are commonly used in models:

- *data* - things to be processed
- *processes* - things performing actions on data, whose definition specifies how to process data
- *processors* - things to carry processing.

Most models allow objects to belong to classes, for example similar object occurrences belong to a named object type. But the modelling rules normally say that each object and equivalently for brevity each object type is a member of only one of the classes: data, process, processor. Thus normally performing a process on another process or on a processor (as opposed to carried out by a processor) is not allowed in such modelling. This seems widely accepted despite the obvious

advantages in compilers and interpreters, in such languages as LISP and PROLOG, and in more classical domains such as mathematical functions.

The separate treatment of the above types stems from a hypothesis that most of the current system design methods are based on. The hypothesis states that the effect of process structure may be omitted for data design. In other words, it has been believed that relationships between data and processes are more or less of the first-order type and a final design of a system might be achieved by applying linear superposition. This approach can only be justified in the context of another (presently widely accepted) 'sequentiality' principle - restricting any process within the target system to being algorithmic, not being interrupted during execution and not having any parallel subprocesses as its own components. Whereas the sequentiality constraint simplifies system design techniques, it can no longer be considered acceptable. In the Real World many processes happen concurrently. In contemporary systems multiple processors may perform many concurrent actions that can interact with each other. These facts should be both anticipated and used to design the elements of the target system, allowing for their better cooperation.

Also, even in systems with a single processor, the structure and sequencing of processes need not be the same as their Real-World equivalents. For example an insurance company's computer system model should allow receipt of and initial partial processing of a claim before the completion of the process of policy renewal and premium payment clearance through banks, and generally wherever the computer sequence may differ from the Real World sequence.

Despite the fact that data analysis techniques are well established and much research effort has been undertaken to develop them, they still show a number of weaknesses. Even the most mathematically sound normalisation and relational approach does not seem to be fully satisfactory.

The insufficient formalism of data semantics is quite commonly recognised but there

are four other problems worth mentioning.

The first one, ***over-abstraction*** resulted from a simplistic use of the Universal Relation concept and too mechanical application of a normalization procedure. All the required attributes are just grouped into one relation with no regard as to their meaning or proper domain definition. Mechanically applied normalization may produce a number of formally correct relations that are then identified with entity types. The point is that these "entities" do not necessarily represent objects from the Real World. They may not possess certain properties required by the relation structure. Hence NULL-values may be needed or some properties which their real world equivalents do possess may not be recordable in that relation. In other words, the relation contains separate groups of tuples, with each group representing some object type. The objects differ from each other despite certain similarities.

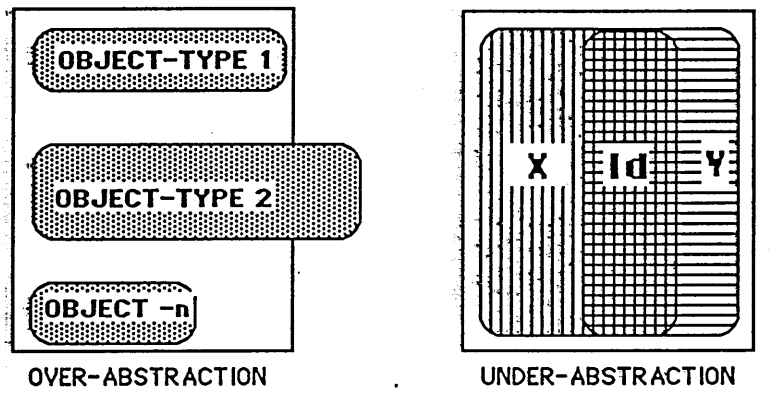



Fig. 1.1

Consider for example a relation that describes bridges. Naturally, some attributes (e.g. length, width, number of spans, capacity) are common for all types of bridges. However, there are attributes applicable only to a particular types as all other types do not have the relevant properties (e.g. cable characteristics apply only to the

suspended structures). A relation that contains all of these attributes would be an over-abstraction, that is, it would be an attempt to represent all types by a single entity.

The second problem may be called ***under-abstraction*** and so far as a single processor environment is concerned it does not normally cause any further trouble. The problem originates from normalization not being powerful enough to represent and act on facts such as that two separate entity types **R'(Id,X)** and **R''(Id,Y)** are being represented in one relation **R(Id,X,Y)** since neither of the functional dependencies X-->Y and Y-->X hold in **R.** Consider for instance a relation:

**Road_Section (Section-name, Administrative-parameters, Traffic-density).**

This relation represents two different objects, namely organizational responsibility for maintaining a number of road sections and a traffic space. Such situations are semantically undesirable and may cause delays in processing retrievals and maintenance routines.

Thirdly, an ***over-categorization*** may occur. This, in some sense, is a problem complementary to those already mentioned. It stems again from too general assumptions made at the meta-level and corresponds to the well-known problem of 'false implication' in Artificial Intelligence:

$$\begin{array}{llllll} \text{X has\_property Y} :: & \text{x} & \in & \text{X} & \rightarrow & \text{x has\_property Y} \\ \textbf{birds fly} :: & \textbf{penguin} & \textbf{is\_a} & \textbf{bird} & \rightarrow & \textbf{penguin flies} \end{array}$$

The clue here lies, of course, in the fact that the statement 'X has_property Y' is not always true. Actually, some elements x belonging to the category X do not have such property, but this peculiarity has been conveniently omitted for the sake of generality. It does not necessarily means that such modelling is erroneous as this kind of speculation may very often be justified as being helpful in most cases. For example a function may be called peculiar at a certain point, meaning that at that

point the function does not have some property otherwise considered general.

The usual technique of resolving this semantically undesirable situation is to decompose X into two sub-categories X' and X'', where:

$X' = \{ x' : x' \text{ has\_a\_property } Y\}$

$X'' = \{ x'' : x'' \text{ has\_a\_property } \sim Y\}$

Y is the complement of ~Y,

$X' \cup X'' = X, \quad X' \cap X'' = 0$

Sensible as it appears to be, continuous decomposition may lead to a combinatorial explosion of categories, e.g. **FLYING_BIRDS** and **NON-FLYING_BIRDS**, each of them being either **SWIMMING_BIRDS** or **NON-SWIMMING_BIRDS**, each of them being either **CARNIVORES** or **NON-CARNIVORES**, and so on. The real world does not have an inherent structure that implies the correct hierarchy or other structure.

A fourth problem, ***infra-abstraction***, arises from the rather serious criticism that the relational notation cannot express certain types of facts and constraints. In particular, relationships amongst elements coming from various (meta)levels of abstraction cannot be expressed at all.

## 1.2 Outline of the thesis and claimed contribution

This thesis consists of ten chapters followed by the list of references and a table of symbols used throughout this present work.

**Chapter 2** introduces the 'integrality' hypothesis stating that an integral treatment of the three aspects (i.e. syntax, semantics and behaviour) is needed in order to describe an information system fully. The objectives of the research then follow.

**Chapter 3** introduces the sentential notation - a simple structured language. Linguistic expressions over this language are called constructs. They are meant to represent objects from the Real World. Every relevant distinguishable part of the Real World is said to be an object. Therefore facts, entities, relationships, relationships between relationships, processes, boundaries and the like are all objects.

Correspondingly, the constructs are made uniform to the extent that their actual form depends neither on their meaning nor on their behaviour. Moreover, it is tacitly assumed that every object is representable as a process. Every construct not actually acting (or being acted upon) is then a declaration (i.e. a static description of a collection of properties), whereas every action performed in the system is a process application. The properties of these actions are investigated in the subsequent chapters without referring to the physical realisation (electronic, mechanical or human) of processors.

The main purpose of introducing sentential notation was to devise a flexible device for describing the Real World - a device close enough to a natural language but without its disadvantages such as ambiguity and inaccuracy. Note that the sentential notation can be seen as some sort of generalisation of the binary relation model introduced by Abrial - but with a much more relaxed structure. Similar approaches may be observed in Deep Structure Elementary Sentences or in Triple-Based Stores.

**Abrial** [1] assumes a basic structure composed of *categories* (such as PERSON) and *bi-directional access functions* (e.g. IS-OF-SEX) that define links between pairs of objects belonging to these categories. A hierarchy of structures can be built up whenever more than two objects are to be associated. To complete the model, five basic operations are introduced (*update, erase, recognize, access, test*). Semantics of data is carried by retrieval programs, i.e. programs that represent questions to the data model.

The term **Triple-Based Stores** [60] refers to a model of data for the storage of *triples* of the form: <*object predicate object*> together with three basic operations *insert*, *delete* and *retrieve*. It is an extension of the binary relationship model to include a number of entities and relationships with predefined semantics (e.g. IS-A relationship). Classes and relationships are viewed as entities at the meta-level. At the top of the hierarchy is the most general class called *thing*; other classes with predefined semantics are lower down with user defined classes lower still. Property inheritance from class to subclass and inference and constraint rules (expressed in clausal form) support semantics of data.

**Nijssen** [63] introduces a concept of *Deep Structure Elementary Sentence* as a *predicate* (e.g.WORKS-FOR) with an ordered set of tuples each consisting of: the kind of thing (e.g. EMPLOYEE), the kind of naming convention (e.g. WITH-EMPLOYEE-NUMBER) and the name of the specific thing itself (e.g. E8856). Nesting is included in these sentences which means that the entire sentences can function as things.

The sentential notation generates wider classes of expressions and seems more natural in describing the Real World. The sentential expressions have a functional form and this makes it possible to apply a denotational style of semantics.

**Chapter 4** outlines a model of semantics seen as a formal system linking up the Universe of Discourse, a syntactic representation of that Universe and one or many Environments of Comprehension (Eoc). The concept of Eoc for semantic description is an original idea. More than one Eoc can be associated with the Universe of Discourse, each of them reflecting a particular view held by a particular group of people. Their views may or may not overlap (partially or totally) and hence their corresponding Eoc may or may not be disjoint. All Eoc are, however, disjoint from the Universe of Discourse and from its representation.

In the context of sentential notation the rigorous denotational approach seems more appropriate than various less formal treatments of data semantics - such as, for instance, semantic networks [73] or other approaches of a more descriptive orientation [74, 77, 85, 95]. Here the meaning of a construct is associated with a function mapping from a set of syntactic constructs to the appropriate semantic domains. Eoc comprises all these domains.

The model of semantics devised in this work can be seen as an extension of denotational semantics for programming languages [13, 88] to cover the issues of modelling data and processes.

**Chapter 5** presents a model of time used subsequently to investigate temporal relationships between process applications. This model of time is, to some extent, a synthesis of various concepts discussed in the relevant literature. For example, **Anderson** [3, 4] describes an axiomatic time model composed of completely ordered set of indivisible time elements (each having a duration). Mappings of each element and each pair of them into the positive real numbers are assumed, and so is time-telling relation yielding a current time.

**Schiel** [76] considers time space as a (possibly complex) structure of intervals (i.e. *continuous and connex* sequence of time points) and defines three primitives:

*interval, before* and *after* to build up an algebra of generalised intervals. Events (widely understood to include operations, transactions and queries) trigger actions within the system subject to the logical value of explicitly expressed predicates, which being an integral part of an event, specify under what conditions (e.g. direct or relative time reference) the event becomes true.

Similarly, **Bubenko** [20] considers a time model as a system composed of an infinite set of *time points* (e.g. real numbers) and a finite, partially ordered set of *time interval types* (such as second, day, month, year). A particular time interval belongs to only one type. Both entities and events have explicitly recorded time references - *existence conditions* and *occurrence conditions* respectively.

A rather mechanistic view is presented by **Richter** [72]. This model relies on (possibly many) time originating devices - directly implemented in the system *pulse generators* in the form of so-called clocks. Each generated pulse is considered a point in time, while two consecutive pulses constitute a basic time interval. For a particular system many 'times' can be designed - at least one *system time* and a number of *regional times*. Petri Nets are envisaged to describe possible mappings between system time and real time and, indeed, between regional times.

The formalism employed in this work differs in the sense that a more rigorous mathematical apparatus was used. The fundamentals were generalised and reformulated to give a cohesive homogeneous system, free from peculiarities of specific time systems that arose from either a specific point of view (e.g. relative or absolute view, time quantum versus time point view) or were consequences of a particular operational framework. An attempt was made to define a general time generic system in the form of relative time-frames, in much the same way as cartesian space coordinates can be seen as inertial frames in the physical space. Particular points worth noting are: separation of the time model as such from a naming convention (such as a calendar) which is a language *per se* with its own

syntax and semantics, precise definition of relationships between any two time sub-systems and a generalisation of a time extracting function.

This work has primarily been oriented towards the problem of system behaviour, behaviour being identified with actions in the system represented by process applications. The formalism developed in this work and described in Chapters 6,7 and 8 is entirely original. It differs, to a degree, from all models previously known to me that describe parallelism of processes [5, 17, 28, 48, 49]. In particular it differs from that used in the theory of Communicating Sequential Processes (CSP) [18, 35, 36, 37, 38, 39]. Although some concepts are similar a few fundamental differences exist: they arose from the different orientation and area of applicability.

To begin with, CSP deals with communication as such in asynchronous parallelism. Therefore, the accent is put there on possible cases of interference of otherwise sequential processes.

Secondly, the orientation of CSP is clearly towards programming languages rather than modelling objects that represent data or information about the Real World. As such, CSP deals with various problems, as for instance buffering, monitoring, pipe, protocols, and resource sharing which are outside the scope of my research.

Thirdly, in CSP the correspondence to (unquantified) time space is rather tentative and no specific model of time is investigated in detail.

Finally, some primitive concepts are differently understood. For example an event in CSP is treated as an element of a process (that is a process may participate in an event) and therefore a process is represented by a trace, i.e. a finite sequence of events the process has engaged up to some moment in time. In the formalism described in this research an event is a kind of fact that triggers (or is a result of) a process application.

These circumstances (and perhaps the fact that CSP was not available in its present form when this research started) made me to develop an alternative approach - surely less comprehensive than CSP but perhaps more suitable for information system analysis and design.

**Chapter 6** explains in detail what exactly is meant by a process application. To do so the idea of a Turing-machine (dealing with only one process at an instant) was generalised to a form giving unlimited simultaneous processes. The abstract machine with two types of channels ensures this and, moreover, it allows one to describe self-modifying and self-regulating systems.

**Chapter 7** investigates temporal relationships that may occur between any two process applications - *strict sequence, inverse sequence, partial sequence, strict parallelism, free parallelism and mutual exclusiveness.* These relationships are formally defined with reference to the time model introduced in Chapter 5. Properties of the above relationships are thoroughly considered in terms of predicate calculus.

**Chapter 8** defines process calculus, i.e. a formal system composed of sentential expressions and logical and temporal operations. The rules for handling expressions are formally proved and so is the theorem concerning precedential completeness. As a consequence, the principle of structured process design is logically deduced.

**Chapter 9** examines the impact of process structure on the relational data model. It has been shown that, in general, the normalisation theory based on functional, multivalued and join dependencies is not capable of detecting certain anomalies in data model; these anomalies may occur due to rejection of immunity of the data model from process structure. An *access anomaly* and a *store anomaly* are identified and described. Furthermore, by considering basic precedence relations (sequence, parallelism, mutual exclusiveness), two new dependencies - *tangled process*

*dependence* and *flat process dependence* are formally defined. Just as functional, multivalued and join dependencies reflect static semantics of data, the process dependencies reflect behavioural properties of process application. Finally it is shown that normalised (BCNF) relations, for which the above dependencies hold, must be decomposed further to avoid access and store anomalies.

**Chapter 10** concludes this thesis by presenting a discussion of how, in my view, the objectives were achieved and listing some suggestions for further research and development.

To summarize, the claimed contribution to the field of information system development include:

- identification of several weaknesses in present methodologies. These weaknesses had not previously been pointed out in the form presented here.

- submission of a hypothesis of the integral treatment of the three aspects of modelling - syntax, semantics and behaviour.

- generalisation of the binary relation model by introducing a sentential notation that is more flexible than previous attempts in this field and that generates a wider class of expressible facts.

- a complete and rigorous formulation of the model of time in the form of interrelated time-frames with three primitive operations and independent from any particular naming convention.

- recognition that time is an inseparable property of information about the real world and making an appropriate provision in the sentential notation to reflect this fact.

- introduction of a new model of semantics of data in the form of a system linking up the Universe of Discourse, its formal syntactic representation and one or many Environments of Comprehension.

- development of a model of parallelism of processes with several kinds of temporal relationships. The model (based on a generalised abstract machine capable of representing actions of multiple processes) comprises formal proofs for various properties of these relationships such as associativity and transitivity.

- development of process calculus with formally proven conversion rules for process decomposition and integration and a formal proof of its completeness.

- investigation of the impact of processes on the data model. In consequence, access anomalies and process dependencies were identified and two new 'normal forms' were suggested, orthogonal to those arising from the concept of data dependence.

Other contribution, of a perhaps less formal nature, includes several arguments in favour of constructive use of parallelism for system development rather than constructing devices to protect the system against undesirable side-effects.

# 2 OBJECTIVES

The fundamental hypothesis we want to base our considerations on originates from Artificial Intelligence studies on natural languages, i.e. on languages currently being researched as most suitable to express the knowledge of a Real World [44, 59, 95]. My hypothesis is that an information system (or for that matter any system or any part thereof) may be fully described only when its three aspects: syntax, semantics and behaviour (pragmatics) have been specified, and these have to be treated integrally.

The following meaning of these 3 words has been taken from C. Morris[62] :

***syntax*** - *deals with the combination of signs without regard to their specific signification or their relation to the environment in which they occur*

***semantics*** - *deals with the significance of signs in all modes of signifying*

***behaviour*** - *deals with origins, uses and effects of signs within the environment in which they occur.*

In this context, the objectives of this work are to:

- develop a notation that is both suitable for the description of information systems and is free from concurrency constraints

- discover the rules governing decomposition (integration) of processes within a target system

- investigate possible effects of process design, being performed integrally with data design, on the data model that represents the Real World within the target system.

These objectives may be regarded as an attempt to extrapolate the principles of structured design into the situation where processes are performed not necessarily sequentially and processes may interact with each other at any point in time. In particular, any process may have subprocesses as components, and these may or may not be performed sequentially.

In this work we shall concentrate on the problem of system behaviour. The treatment of syntactic and semantic issues will be limited to those aspects that are needed in order to explain the problems of system behaviour.

# 3 BASIC ASSUMPTIONS AND DEFINITIONS

The assumption we start with is that the relevant part of the Real World is discretisable, i.e. may be modelled by a structure of *objects*. We do not distinguish anything other than objects. The word *object* is to be interpreted in a wide sense so it can mean *anything* within the boundaries of the part of the Real World that is to be represented in the target system (hence, in particular, *a structure* and *a boundary* are *objects* too and so is a process, an event, an information flow and time). Thus an object may denote a fact or a physical or abstract entity, or relationships between any of these, or relationships between relationships, or any even still more complex structure.

The syntactic representation of an object within a target system is called a ***construct.*** A construct that cannot be decomposed further (either syntactically or semantically) is defined to be an ***atom***. Any construct is a result of the following syntactic production rules, written below in the BNF convention:

(3.1) construct ::= atom | function | function construct
function ::= atom | (construct)

The first rule means that a construct is either an atom (not meaningfully decomposable), or a function followed by a construct. The first alternative means that every atom is a construct, while the second defines the left-side construct as the effect of the function operating on the construct that follows it. This association to the left is not significant, i.e. it does not matter that we have chosen functions to be prefixes rather than suffixes.

At the level of conceptual modelling that is the whole syntax that was assumed. It forms ***a sentential notation***. We choose to call it sentential for its resemblance to sentences in natural languages. Its structure has been based on that of the *lambda-notation* [23].

The sentential notation has certain advantages: while simple yet powerful it allows for denotational style of data semantics to be applied as the meaning of a construct may be derived from the meaning of its components. It is not an original concept; in fact Deep Structure Elementary Sentences [63], Binary Relationship Approach [1] and Triple Stores [60] are all very similar to it. The difference however appears in the wider class of expressions generated by the sentential notation, and in particular in the fact that it explicitly allows for constructs such as **atom atom** and **function function.**

The constructs within the target system may represent four *kinds* of objects: ***facts, events, process_declarations*** and ***process_applications***. The definitions of these objects appear below:

**FACT:** A formal representation of an assertion, rule, event, condition, relationship or function. Facts can be either simple i.e. referring to the same level terms, or generalised (i.e. fact-formulae) containing terms from different (meta-) levels. Facts may be connected by functions such as logical functors (and, or, not) or associative connectives (e.g. is_a, is_of, has_a_property)

**EVENT:** A kind of fact within the system that causes an action , e.g. at an event a process may start or finish or continue or be interrupted. In general, we say that an event triggers a *process application*; an event may trigger more than one process application and, conversely, a particular process application may be triggered by more than one event.

**PROCESS_DECLARATION:**

A static description that consists of facts specifying:

- events that trigger the process application
- constructs to be taken as inputs
- constructs to be produced
- production rules
- events caused by process application
- preconditions and postconditions
- rules about parallelism

**PROCESS_APPLICATION:**

An action performed by a processor where a process is activated in order to process a construct; at the moment of activation the process declaration becomes a 'control image' for the processor to carry out the task.

We assume that **all processors** within the target system are universal, i.e. may perform any process application appropriate for that system, unless specified otherwise. We aim to understand the logic of these actions without reference to the actual physical realisation of the processors.

Summarising the above definitions we may typically say that **whenever at a time T an event E occurs, a process P is applied on a construct C producing a construct C' and causing an event E'.**

# 4 MODEL OF SEMANTICS

The main purpose of this work is to investigate the problem of system behaviour, each system being modelled as a number of somehow structured processes that cooperate in some way to process syntactic constructs. These constructs represent objects existing in the Real World. Objects, in turn, may be physical or abstract entities, concepts or phenomena, or processes themselves.

It must be stressed here again that the semantic aspects of system description are presented in this work in a very much simplified form and only to the extent required to understand better the behavioural issues. A more complete and precise treatment of semantics would require further work which it would not have been possible for me to research fully.

The formal representations of objects have been, to a great extent, made more uniform by adopting simple syntactic rules (3.1). Therefore a suitable semantic apparatus needs to be deployed to determine the meaning of a construct and to be able to distinguish different constructs from each other, in view of their possibly similar form.

Below, a model of semantics (within the context of information systems) is outlined. It is an original approach, though the influence of the Scott-Strachey style [88] appears. Some connections to INFOLOG's philosophy [79, 82, 89] and the Montaque-like treatment of semantics for natural languages [14, 59] also appear.

So far two separate structures - the part of the Real World that is of interest and the corresponding formal model - have been distinguished. From now on we shall refer to that part of the Real World as the **Universe of Discourse** (**U**$_{OD}$). It contains all those objects of interest *'that have been, are, or ever might be in the Real World'* [33]. Thus the **U**$_{OD}$ is assumed to be time-dependent. At a certain instant a particular object may or may not be 'available', i.e. at that instant this object may or may not be an element of the **U**$_{OD}$.

The objects in the **U**$_{OD}$ are assumed to be of either of two kinds - atomic or composite. **Atomic** means not meaningfully decomposable while **composite** means decomposable into a number of other objects, each of which may again be either composite or atomic. The set of all atomic objects is denoted by $\{u_a\}$. Similarly $\{u_c\}$ denotes the set of all composite objects.

Among the atomic objects the veridicity set :

$$\mathbf{V} = \{\text{concept of truth, concept of falsehood}\}$$

plays a particular role, which will be investigated later in this chapter.

Thus

$$\mathbf{U}_{OD} = \{u_a\} \cup \{u_c\}$$

$$\{u_a\} \cap \{u_c\} = 0 \qquad (4.1)$$

$$\mathbf{V} \subset \{u_a\}$$

Correspondingly, there is a formal system of constructs. The set of all such constructs, i.e. the model **C** is composed of both atomic constructs $\{c_a\}$ and composite constructs $\{c_c\}$. The production rules (3.1) can be regarded as criteria of memebership. The subset of $\{c_a\}$ corresponding to **V** in the Universe of Discourse is called **Bool** = {true, false}.

$$\mathbf{C} = \{c_a\} \cup \{c_c\}$$

(4.2) $$\{c_a\} \cap \{c_c\} = 0$$

$$\mathbf{Bool} \subset \{c_a\}$$

Note that in the context of (3.1) we do not distinguish atoms from atomic functions. In the model presented here we assume that an analyst may merge atoms representing objects (such as JOHN or BOOK) and atomic functions (such as LIKES or HAS). The question whether a particular construct plays the role of an object or a process will be determined by its position in the relevant expression and by the rules of system behaviour.

Every object (whether atomic or composite) has a description. The descriptions are such that any two objects are distinguished. A description itself is not an object from the **UoD** , nor is it a construct from **C**. For instance an object $\Omega \in$ **UoD** has a description: *'Stefan Stanczyk, the author of these thesis'*; the corresponding construct in **C** may be of the form *'S. Stanczyk'*. Thus a description is said to define a subset of all those objects of the **UoD** that conform to its essence.

In constructing a model for semantics we are looking for a correspondence between formal syntactic constructs and appropriate objects from any UoD. We aim to associate the meaning of a construct - or more precisely its abstraction - with a function that yields a value from a specific semantic domain. The concept of identifying that abstraction (rather than directly its value) with the meaning ensures that each construct is given a stable meaning - free from a particular context and independent of the particular UoD instance.

The set of all abstractions of these functions constitutes the **Environment of Comprehension** (EoC). Note that we do not require that for any given Universe of Discourse a single unique EoC should be associated with it. In general, one or many EoC can be devised, each reflecting a particular view of the UoD. This is yet another design decision to be made by the system analyst. Depending on circumstances, various EoC may or may not be disjoint with each other. Each EoC is, however, disjoint with both UoD and C. It is an image of how a different group of people understand various phenomena (believed to occur in the Real World objectively), quite irrespective of the possibly different notations that they may use to record their understandings. For instance the Environment of Comprehension of people having knowledge of nuclear physics differs from that of those who do not have, if both groups were to describe an atomic explosion. Similarly, the EoC of system designers possessing certain technical knowledge - e.g. of computer programming and processing - is very different from the EoC of those knowing only manual means of processing.

The model of semantics requires an appropriate formal system to be devised. That is, my aim is to develop and describe a system linking in some way the three concepts - UoD, C and EoC. Fig. 4.1 below illustrates this idea.

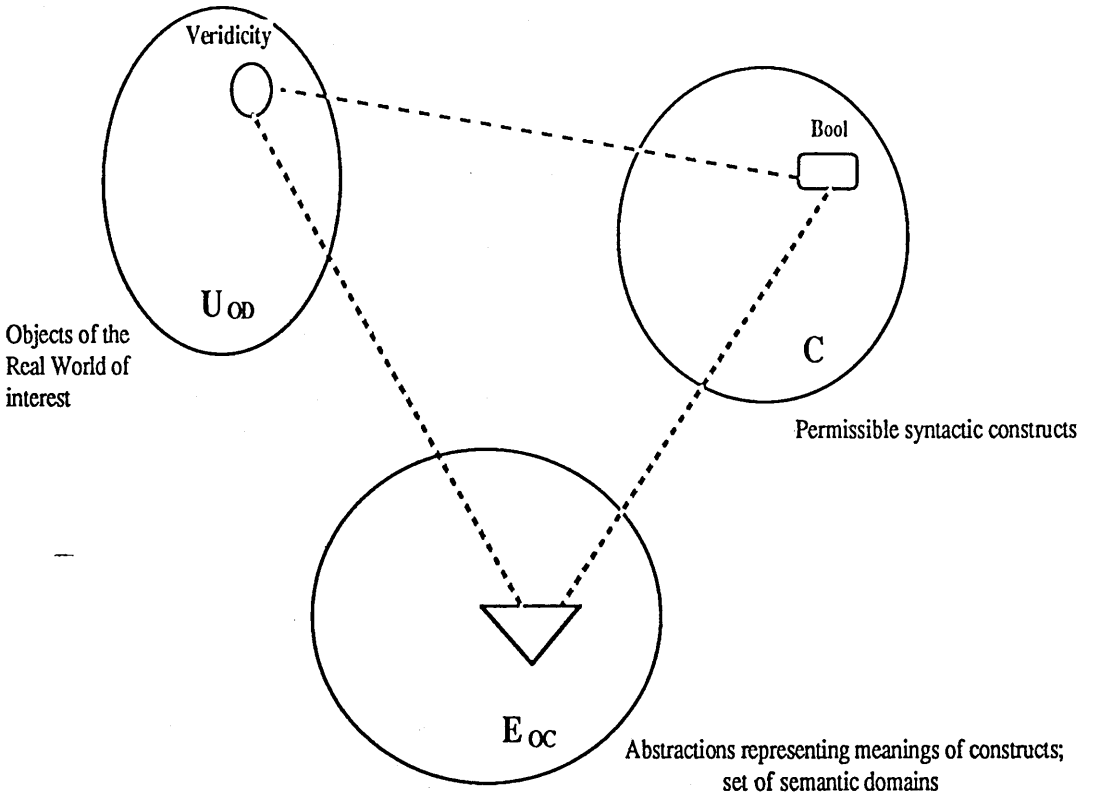



Fig. 4.1

The resemblance of this concept to the approach demonstrated in [33] for Information Systems is deliberate. We aim to adopt the three basic principles stated there, but with some amendments as explained below. These principles, translated into the terminology used here, may be expressed as follows, with square brackets round my changes.

'1. [C ] is a formal system, *the [Eoc] as a whole is not so*

2. The behaviour of [C ] is completely defined by behaviour rules and constraints, which are established, directly or indirectly, by the [Eoc]. [C ] on its own initiative never establishes rules for the [Eoc].

3. [C ], being fully predictable, is unable to deviate from the rules or constraints, *[Eoc] can deviate from its rules.* '

The parts printed in italics, will be rejected for the reason that, indeed, a formal system for [Eoc] is being sought. More precisely, we are to limit our consideration to that part of the Environment of Comprehension that can feasibly be formalized - just as we did by imposing certain restrictions (e.g. the discretization principle) on the Real World. Those restrictions excluded some Real World's objects from the Uod.

As we have said earlier, each Uod consists of two kinds of objects - atomic and composite. Similarly, C comprises two kinds of constructs, also termed atomic and composite. Relationships among these kinds of objects (depicted by arrows in Fig. 4.2) are assumed to define the structure of Eoc; hence an analysis of these relationships is needed.

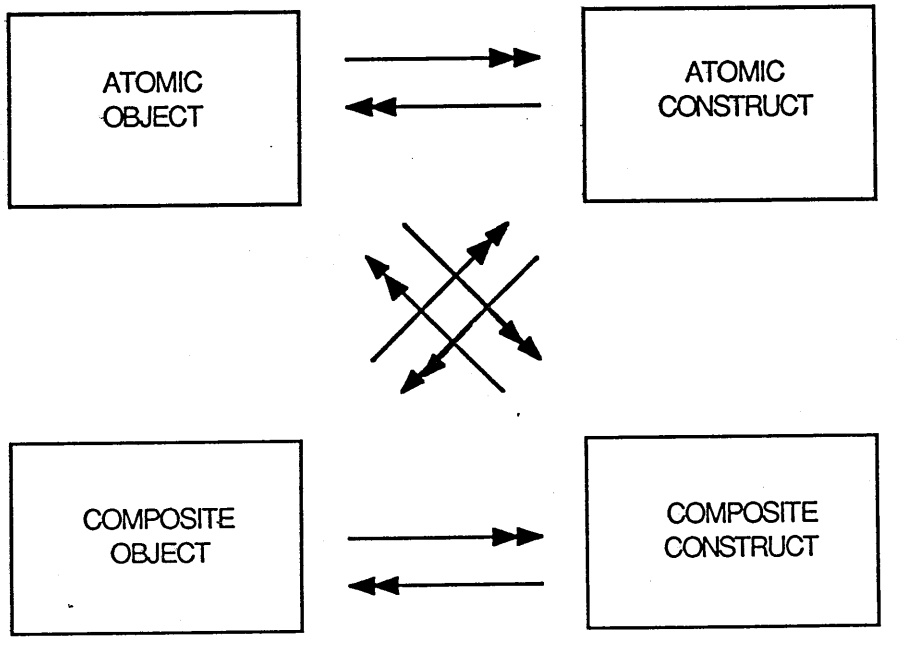
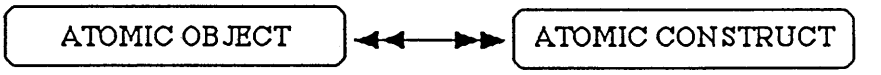



Fig. 4.2

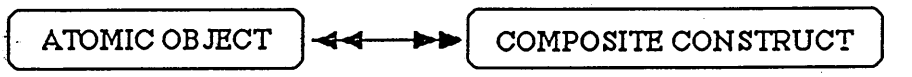
## RELATIONSHIPS BETWEEN OBJECTS AND CONSTRUCTS

There is a many-to-many relationship between atomic objects and atomic constructs.

**Relationship :** 


This is the most fundamental type of relationship. It says that for every atomic object there exist a corresponding atomic construct; more than one such construct may represent a single atomic object (i.e. synonyms are allowed). Conversely, one atomic construct may refer to many atomic objects. This property, questionable in many formal systems, is just a direct observation coming from existing applications. In various databases, for instance, a linguistic construct 'Smith' may denote a number of people; similarly the formal construct '0' may refer to either the number zero or a concept of falsehood. This observation stresses somehow the necessity of having proper semantic definitions (the ambiguity itself is often referred to as the *'weak semantic barrier'*).

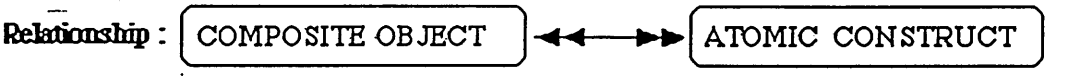
There is a many-to-many relationship between atomic objects and composite constructs

**Relationship :** 


Some atomic objects do not have direct representation in the form of atomic constructs. In such cases therefore a composition of atoms should be needed. This composition may be formed in a number of ways, resulting in a number of semantically equivalent constructs (i.e. 'different descriptions of the same thing'). The constructs may be reducible to some canonical form (e.g. 00127 = 127), but in many cases this is (intentionally!) not so. An illustration of the latter is the enumeration system where a number has infinitely many numerals associated with it, as for instance $11001_2 = 221_3 = 121_4 = ... = 25_{10} = ...$

Conversely, a particular composite construct may denote different atomic objects (*'strong semantic barrier'*) depending on what interpretation (i.e. meaning) is applied to that construct. Drawing an example again from number theory, a linguistic construct 10011001 produces 153 when simple binary evaluation is carried out; on the other hand the function **binary_coded_decimal** will produce the denotation 99.

A many-to-many relationship between composite object and atomic construct exists:



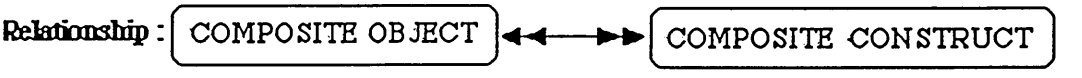

Very often composite objects are given an atomic representation in C, particularly in situations where hierarchies or other structures are being made out of many atomic objects. But a particular construct does not have to refer to a single composite object. For instance, the construct *'Theory'* may correspond to *'A coherent system of objects that comprises description of relevant phenomena , conceptualization techniques, assumptions, criteria and mathematical formulae'*. The same atomic construct may, however, refer to *'an object having no direct relationship to practice'*.

Again, a composite object may have many representations in C. For instance in modelling roads an object

*STRUCTURE(Geometric Shape, Construction, Traffic Capacity)*

can be referred to as *Section* or *Link* or *Element*.

A relationship between composite object and composite construct exists:




It follows from the above considerations that this is a many-to-many relationship.

The Environment of Comprehension contains semantic domains, that is sets of functions that designate meanings for syntactic constructs. The semantic domain comprising all meanings for atomic constructs (i.e. atoms and atomic functions) is called $\mathbf{B} \subset \mathbf{Eoc}$.

The meanings of the most fundamental atoms - the primitives true and false, both belonging to **Bool**, are defined as mappings (using RW for Real World):

(4.3) $\mu$: true $\rightarrow$ RW concept of truth

$\mu$: false $\rightarrow$ RW concept of falsehood

The possible abstractions of $\mu$ do not necessarily have to be equivalent in the sense of the axiom of extensionality. For instance these abstractions may take the following form:

(4.4) $\lambda x \lambda y.x$ for truth

$\lambda x \lambda y.y$ for falsehood

But the Real World concept of veridicity may as well be expressed by the Aristotle's phrase [2]:

> *'To say of what is that it is not, or of what is not that it is, is false, while to say of what is that it is, or of what is not that it is not, is true'.*

Hence in general, the meaning of an atomic construct is defined by:

(4.5) $\mu$: $\{c_a\} \rightarrow \{\mathbf{U_{OD}} \rightarrow \mathbf{Bool}\}$

The function $\mu$ draws values from a set of functions $\{U_{OD} \rightarrow B_{OOL}\}$; this approach guarantees a freedom of choice from many possible descriptions, only one of them to be accepted in $E_{OC}$.

With respect to the denotational principle [88], saying that the meaning of a construct must be obtainable from the meaning of its components, the meaning of a composite construct may be defined as:

(4.6) $\eta$ (construct atom) = $\eta$ (construct) $\mu$(atom)

Fig. 4.3 illustrates in some detail an example of a model of semantics suitable for natural numbers.

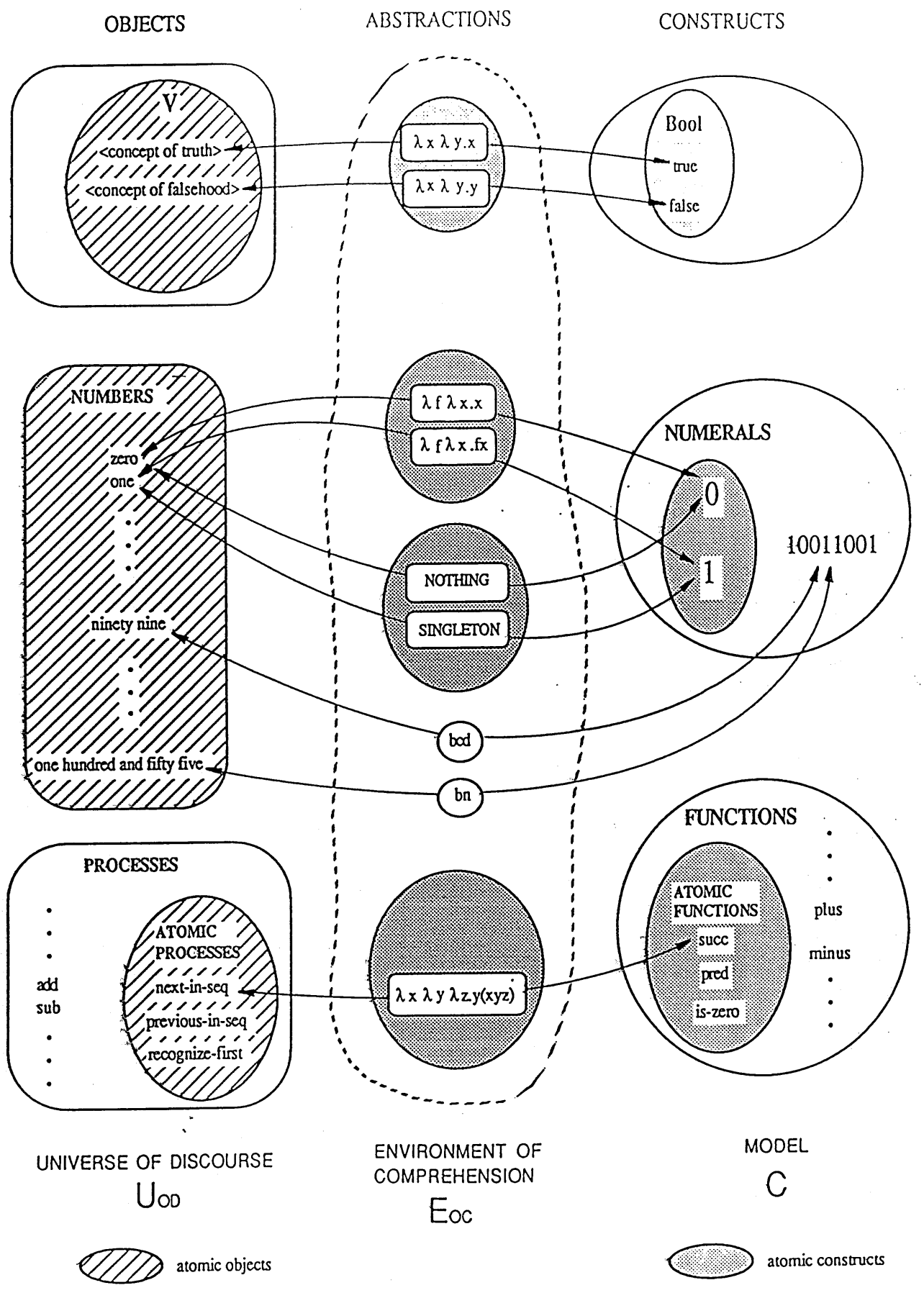

OBJECTS
ABSTRACTIONS
CONSTRUCTS
V
<concept of truth>
<concept of falsehood>
λ x λ y.x
λ x λ y.y
Bool
true
false
NUMBERS
zero
one
ninety nine
one hundred and fifty five
λ f λ x.x
λ f λ x.fx
NOTHING
SINGLETON
bcd
bn
NUMERALS
0
1
10011001
PROCESSES
add
sub
ATOMIC PROCESSES
next-in-seq
previous-in-seq
recognize-first
λ x λ y λ z.y(xyz)
FUNCTIONS
ATOMIC FUNCTIONS
succ
pred
is-zero
plus
minus
UNIVERSE OF DISCOURSE
UOD
ENVIRONMENT OF COMPREHENSION
EOC
MODEL
C
atomic objects
atomic constructs

Fig. 4.3

# 5 MODEL OF TIME

Temporal aspects of modelling are extremely important, particularly in a situation where concurrency is considered as a natural property of certain classes of objects and not merely an abstract mechanism to tackle problems of communication between otherwise sequential processes. Therefore the issue of time modelling is given a separate chapter.

There is an equivalence between temporal and spatial aspects. Any object in the Real World may be referenced to by time-coordinates as well as space-coordinates. Indeed, just as Cartesian coordinates locate an object in some physical space, so do the time-coordinates in the time-space. The only difference stems from the ability of an observer (or an object) located in the *physical space* to move in any direction, whereas in the *time-space* we assume no such a movement by an inside-observer (or an object) is feasible.

A similarity and an important distinction have to be noted concerning space and time. Objects have some properties expressible in time units (e.g. BRIDGE # 123 **is_of_age** 35 YEARS) and others in metric units (e.g. BRIDGE # 123 **is_of_length** 250 METRES).

The point is that time as a domain for attribute values needs no separate treatment from any other domain, be they PERSON_NAMES, PART_NUMBERS or TRAFFIC_DENSITY. Generally time *as a concept of reference* does however require special attention.

In our considerations we shall be looking for an appropriate mechanism to express both simple precedences and more complex functions required to locate an object in time. The simple precedences state merely that something must have happened before something else may take place; however an object's time-location may be defined subject to information about other objects, e.g. their locations and the relationships amongst them.

In our investigations time will be considered as a natural and universal attribute of the Real World. We assume the fundamental axiom:

***In the Real World to be modelled there exists***
***an unique and absolute time reference system.***

A possible mathematical representation of the above axiom is that the universal time $\mathbf{T^0}$ is represented by a set **Q** of real numbers $t_1, t_2, \ldots$ with a linear ordering $\leq_t$ and a metric **d** that have the following properties:

**for all** $t_1, t_2, t_3 \in \mathbf{T^0}$ **:**

(5.1)

$t_1 \leq_t t_1$ *reflexivity*

$t_1 \leq_t t_2$ **and** $t_2 \leq_t t_1$ **implies** $t_1 = t_2$ *anti-symmetry*

$t_1 \leq_t t_2$ **and** $t_2 \leq_t t_3$ **implies** $t_1 \leq_t t_3$ *transitivity*

**for all** $t_1, t_2 \in \mathbf{T^0}$ **either** $t_1 \leq_t t_2$ **or** $t_2 \leq_t t_1$

for all $t_1, t_2, t_3 \in T^0$ :

$$d(t_1,t_2) > 0 \quad \text{for } t_1 \neq t_2$$

(5.2) $$d(t_1,t_2) = 0 \quad \text{for } t_1 = t_2$$

$$d(t_1,t_2) = d(t_2,t_1)$$

$$d(t_1,t_3) \leq d(t_1,t_2) + d(t_2,t_3)$$

This abstraction of the time system is rather precise and reflects well the nature of human cognition of time. However, being continous it is too general for our purposes. Also importantly, being independent from the coordinates of an 'inside-observer' does not allow for an object to be located relative to the observer's own position. Nonetheless, we shall retain the continous model of time as a base to which any ***local time system*** can be referred to as shown below.

The notion of a local time system is central to our considerations. Analogously to the cartesian space-coordinates, whenever the conditions require it the local time system may be set up. Essentially it comprises the following elements:

- an infinite, countable set of time intervals
- a relation $\leq_t$ totally ordering the above set
- a metric $d^i$ that defines the length of time
- a precedence-preserving function $e^{ij}$ relating a local time to the global time or to another local time
- a functional device $CLOCK^i$

The formal definitions of these terms appear below.

Both the absolute time system and each local time system can usually be approximated by discrete rational or integer values of time. **A local time,**

$\mathbf{T^i} = \{ \{t_j^i\}, \leq_t \}$ consists of discrete indivisible time elements $t_j^i$. All intervals are made the same length which is the smallest possible unit of time within $\mathbf{T^i}$. The length of an interval, called the **resolution** is incomparably small with respect to the duration of any activity that may take place within the scope of the local time system. Every activity lasts an integer multiple of the smallest unit of time. The resolution is a property pertinent to a particular time; in general the resolution in different local time systems may be different.

**The relation** $\leq_t$ is a linear ordering in $\{t_j^i\}$:

**for all** $t_1^i, t_2^i, t_3^i \in \{t_j^i\}$

$t_1^i \leq_t t_1^i$ *reflexivity*

(5.1a) $t_1^i \leq_t t_2^i$ **and** $t_2^i \leq_t t_1^i$ **implies** $t_1^i = t_2^i$ *anti-symmetry*

$t_1^i \leq_t t_2^i$ **and** $t_2^i \leq_t t_3^i$ **implies** $t_1^i \leq_t t_3^i$ *transitivity*

**for all** $t_1^i, t_2^i \in \{t_j^i\}$ **either** $t_1^i \leq_t t_2^i$ **or** $t_2^i \leq_t t_1^i$

**The metric function** $\mathbf{d^i}$ defines a distance between two time-intervals

**for all** $t_j^i, t_k^i, t_l^i \in \mathbf{T^i}$

(5.2a) $d^i(t_j^i, t_k^i) > 0$ **for** $j \neq k$

$d^i(t_j^i, t_k^i) = 0$ **for** $j = k$

$d^i(t_j^i, t_k^i) = d^i(t_k^i, t_j^i)$

$d^i(t_j^i, t_l^i) \leq d^i(t_j^i, t_k^i) + d^i(t_k^i, t_l^i)$

(5.3) $\mathbf{d^i}(t_j^i, t_k^i) = 1 + \mathbf{d^i}(t^i_{j+1}, t^i_k)$

for the sequence $<\ldots t^i_{j-1}, t^i_j, t^i_{j+1}, \ldots, t^i_{k-1}, t^i_k, t^i_{k+1}, \ldots>$

We may now say that any discrete local time is adequately represented by a set of integers **I**, thus $\mathbf{T^i} = \{\mathbf{I}, \leq\}$ (see 5.1a).

**The function $\mathbf{e^{ij}}$** relates two local time systems $\mathbf{T^i}$ and $\mathbf{T^j}$ to each other in the following way. $\mathbf{e^{ij}}$ maps time points in $\mathbf{T^i}$ to equivalents in $\mathbf{T^j}$. The origin of one local time system, $t^i_0$, will correspond to a particular value in the other, say $t^j_p$

(5.4) $\mathbf{e^{ij}} : \mathbf{T^i} \rightarrow \mathbf{T^j}$

where: $e^{ij}(0) = t^j_p$

There is, of course an element of choice in assigning a specific point $t^j_p$ that a zero-point in $\mathbf{T^i}$ maps into, though this matter is rather secondary. Far more important is the issue how the location of $t^j_p$ in $\mathbf{T^j}$ is specified. This location is normally given indirectly by an associated fact, or a combination of facts which have occurred (or will occur) at that point in $\mathbf{T^j}$.

So long as $\mathbf{T^i}$ and $\mathbf{T^j}$ have the same resolution the function $\mathbf{e^{ij}}$ is bijective and continous, so both precedences and distances are preserved. More often though this is not the case (as for instance when relating the universal time to a local time, with $\mathbf{T^i} = \mathbf{T^0}$). We shall however require for the function $\mathbf{e^{ij}}$ relating the local time to the universal time to be precedence-preserving.

Formally, the function $e^{ij}$ may be defined as follows (see Fig. 5.1b). We are going to map each discrete time *point* in $T^i$ into an corresponding *interval* in $T^j$. The interval will be open at the lower (starting) end and closed at the upper (finishing) end.

**for two sequences:**

$$T^i = <\ldots, t^i_k, t^i_{k+1}, t^i_{k+2}, \ldots>$$

$$T^j = <\ldots, t^j_p, t^j_{p+1}, t^j_{p+2}, \ldots, t^j_{q-1}, t^j_q, \ldots t^j_{r-1}, t^j_r, \ldots, t^j_{s-1}, t^j_s, \ldots>$$

$$\vdots$$

$e^{ij}$ :

$$t^i_k \rightarrow (t^j_p, t^j_q]$$

$$t^i_{k+1} \rightarrow (t^j_q, t^j_r]$$

$$t^i_{k+2} \rightarrow (t^j_r, t^j_s]$$

$$\vdots$$

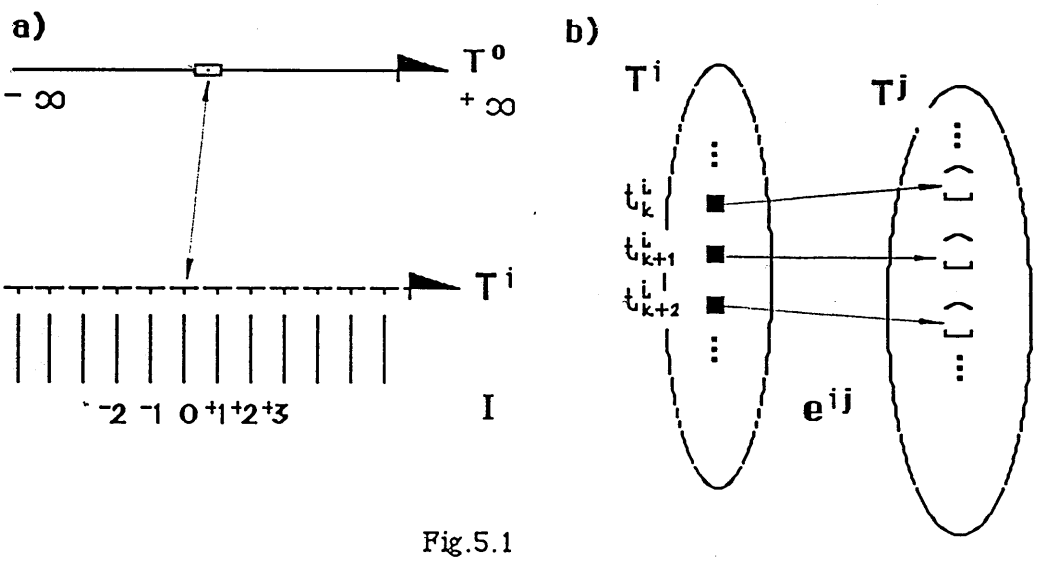



Fig.5.1

Note, that $e^{ij}$ is a function when regarded as a mapping to intervals, since every element $t_k^i \in T^i$ is mapped to exactly one open-closed interval in $T^j$. But it is a mapping *to-many* in the sense that each $t^i_k$ maps to many points within that interval.

The mapping $e^{ji}$ is not the reverse of $e^{ij}$ because the image under $e^{ij}$ is an interval, and the domain of $e^{ji}$ is a time point. Similarly if one converts an integer Farenheit temperature to Celsius to the nearest integer, and then converts back, the result may differ from the original.

The mapping as defined by (5.5) does preserve precedences but not necessarily distances:

(5.6) **for all** $t^i_k, t^i_l \in T^i$ **and** $t^j_p, t^j_q \in T^j$

$$(e^{ij}(t^i_k) \rightarrow t^j_p) \text{ and } e^{ij}(t^i_l) \rightarrow t^j_q)$$

$$\text{implies } (t^i_k \leq_t t^i_l \text{ iff } t^j_p \leq_t t^j_q)$$

(5.7) **there is** $t^i_k, t^i_l \in T^i$ **and** $t^j_p, t^j_q \in T^j$

$$(e^{ij}(t^i_k) \rightarrow t^j_p) \text{ and } e^{ij}(t^i_l) \rightarrow t^j_q)$$

$$\text{implies } d^i(t^i_k, t^i_l) \neq d^j(t^j_p, t^j_q)$$

**A functional device** $\text{CLOCK}^i$ may be associated with every time system $T^i$. It counts the time intervals elapsed since the establishment of $T^i$ and may be thought of as a mechanism giving the actual value of a function 'time_now'. Mathematically it may be represented by a function with no arguments that returns a non-negative integer:

$$\text{CLOCK}^i = \{\} \rightarrow {}^+I$$

We define now three primitive functions **succt** (successor-time), **predt** (predecessor-time), and **att** (at-time). These are basic operations allowed on T and their role is quite analogous to that in the theory of arithmetics. For notational convenience we omit the superscripts indicating a particular time system.

For a sequence

$S = < ..., t_{k-1}, t_k, t_{k+1}, ..., t_m, t_{m+1}, ..., t_n, t_{n+1},...,t_{p-1},t_p,t_{p+1},...,> \subseteq T$

(5.8) $\mathbf{succt}\ (t_k) \rightarrow t_{k+1}$

$\mathbf{succt}\ (..., t_p, t_{p+1}] \rightarrow t_{p+2}$

(5.9) $*\mathbf{succt}\ (t_k) \rightarrow [\ \mathbf{succt}\ (t_k), {}^{+}\infty)$

$*\mathbf{succt}\ (..., t_p, t_{p+1}] \rightarrow [\mathbf{succt}\ (t_{p+1}), {}^{+}\infty)$

(5.10) $\mathbf{predt}\ (t_p) \rightarrow t_{p-1}$

$\mathbf{predt}\ [t_k, t_{k+1}, ... ,) \rightarrow t_{k-1}$

(5.11) $*\mathbf{predt}\ (t_p) \rightarrow (^{-}\infty\ , t_{p-1}]$

$*\mathbf{predt}\ [t_k, t_{k+1}, ...) \rightarrow (^{-}\infty\ , t_{k-1}]$

(5.12)

| | |
|---|---|
| **att** ({}) → undefined (**bottom**) | with the interpretation: "a construct being located by this function never occurred" |
| **att** (S') → S', for any S' ⊆ S | fixed point |
| **att** (T) → undefined (**top**) | with the interpretation: "a construct being located by this function is valid throughout the whole T" |

Given two periods of time $(t_k, t_{k+1}, ..., t_p, t_{p+1})$ and $(t_m, t_{m+1}, ..., t_{n-1}, t_n)$ the function **witt** (within-time) decides whether the first one wholly contains the second:

(5.13)

**witt** $((t_k, t_{k+1}, ..., t_p, t_{p+1}),(t_m, t_{m+1}, ..., t_{n-1}, t_n)) \rightarrow$ true **iff** $(t_k \leq_t t_m$ **and** $t_n \leq_t t_{p+1})$
false otherwise

**witt : T x T → Bool**

In any time system $T^i$ a notation $L^i$ for naming units of time is needed. Such a notation will usually consists of a number of words (such as year, month, day), i.e. terms used as names for some chosen time-units. There is, of course, a need for a convention (a set of syntax rules) to form a name for an arbitrary unit of time. The notation must also be supported by semantic rules that define the mapping $L^i \rightarrow T^i$ for any syntactically correct term from $L^i$.

Consider as an example $L^i$ = CALENDAR$^i$ with a resolution, say **h** (for hour). Though the actual syntax may vary from system to system, the 'values' commonly allowed for are each of the form **y:m:d:h** where:

$y \in I$, $m \in [1,2,...,12]$, $d \in [1,2,...,31]$, $h \in [0,1,...,24]$.

The colons between the letters indicate that, semantically, the representation **y:m:d:h** is a positional one, though due to some physical laws and tradition no common radix (as in numerals for numbers) exists. The time may be evaluated to a unique point on the Gregorian time-scale and, conversely, any point on that scale may be assigned a unique value to the nearest hour. The representation of the value in the form **1986:12:25:10** may be regarded as a named atom.

In this work we shall not be considering any details of $\mathbf{L^i}$. We assume that there exists some suitable convention for giving a name to a time reference and that such reference is an atom in the sense of (3.1), having the meaning in $\mathbf{B_t \subset B}$. Our concern is to provide functions that locate any construct produced by (3.1) in time and that are able to express temporal relationships among these constructs.

The function which for any given construct extracts its time reference is called **when**, defined as follows:

(5.14) $\mathbf{when} = \lambda t.\ \mathrm{construct} : \mathbf{C} \rightarrow \mathbf{T}$

$$\mathbf{when}(E=\{e_1,e_2\}) = \mathbf{when}(e_1) \cap \mathbf{when}(e_2)$$

Of particular importance are three functions: **before, after** and **during,** whose definitions appear below. Their role is to decide - for each ordered pair of constructs taken from C - in what relations of temporal precedence a particular pair of constructs remains.

(5.15) $\mathbf{before} : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{Bool}$

$$\mathbf{before} = \lambda c.\ \underline{\mathrm{if}}\ \mathbf{succt}(\mathbf{when}(c')) \leq_t \mathbf{predt}(\mathbf{when}(c''))$$

$$\underline{\textit{then}}\ \mathrm{TRUE}\ \underline{\textit{else}}\ \mathrm{FALSE}$$

which is a formal expression of the following observations

*if* **when**(c')→ $<t_i, t_{i+1}, \ldots, t_k>$ **and** **when**(c'')→ $<t_j, t_{j+1}, \ldots, t_l>$
*then* **succt**($t_k$) $\leq_t$ **predt**($t_j$)

(5.16) **after** : C x C → **Bool**

**after** = λc. *if* **succt**(**when**(c'')) $\leq_t$ **predt**(**when**(c'))

*then* TRUE *else* FALSE

(5.17) **during** : C x C → **Bool**

**during** = λc. *if* **att**(**when**(c'')) **witt att**(**when**(c'))

*then* TRUE *else* FALSE

# 6 MORPHOLOGY OF PROCESS APPLICATION

So far some issues of representing a selected portion of knowledge about the Real World have been considered. The sentential notation was defined to record this knowledge - the objects of the Real World may now be represented by formal constructs and these may be given meaning in a functional form. The means to locate objects in a certain time-space were also provided.

However, the description of a target system is not yet complete. For, though the discretisation of the Universe of Discourse may have been done suitably and all the objects may have been properly defined, the notations so far lack an adequate means of representing and explaining actions that may take place. Similarly the circumstances (i.e. general rules that govern these actions) have yet to be described.

What we need now is an appropriate model capable of characterizing process applications. Depending on certain conditions, some constructs are to be interpreted as process declarations and will have to be activated whenever conducive circumstances occur. Operationally it means that an idle processor will acquire a specific process declaration which from that moment on will normally control its action until the end of the application. The relevant constructs will be processed yielding new constructs that describe new objects and new circumstances.

We aim to explain these phenomena without referring to any physical properties of processors. In fact, we do not wish to analyse a processor as such at all. Instead, an abstract mechanism will be used to explain the process application by considering the observable results.

There are four preconditions that determine the model of the process application mechanism:

– *First,* it must be rich enough to be compatible with the conventions for syntax, semantics and time-referring - it must, of course, comply with the assumptions already made. In particular, the model must be capable of explaining an application of any **X** to any **Y** irrespective of what the actual interpretation of **X** and **Y** might happen to be.

*Second,* the model must be conceptually consistent with the idea of parallelism. This is where the concept of all processors being universal comes to use. We take a view that any application may be carried out by any processor not actually engaged in some other action.

*Third,* the model must allow for a description of a 'self-modifying' system, i.e. a system whose objects may get changed due to its internal action rather than by external (e.g. designer's) intervention.

*Fourth,* the concept of a 'central control' being compulsory in the traditional model of computing has been rejected. The Real World has no equivalent. We are actually looking for a 'self-regulating' model, where relevant objects of the system are supposed to take the necessary action if and when the circumstances allow for. In other words, a process application occurs 'automatically' as soon as the relevant constructs become available. (The term 'relevant constructs' may also include processors, resources, conditions and controls - not merely 'data').

It does not necessarily mean that the central control is ruled out. If needed, it may be *consciously* implemented by the system designer but in such case the appropriate constructs must be explicitly described in terms of syntax, semantics and behaviour. The system control may, in fact, be a complex structure of processes (e.g. a hierarchy or interconnected network), each part of it coordinating a specific part of the whole Universe of Discourse.

An abstract machine employed to explain the process application is depicted in Fig.6.1. It consists of two input channels and two output channels connected to a 'black-box' processor.

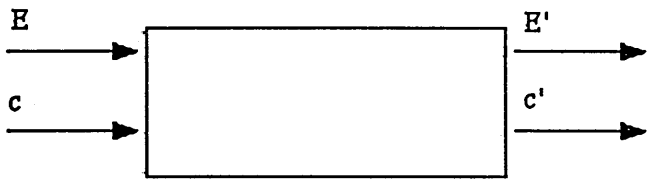



Fig. 6.1

Suppose that the event(s) triggering a particular process application have occurred; the relevant information is immediately received by the first input channel, which selects an appropriate process declaration and transmits that into the processor. The second input channel is then ready to accept an appropriate construct for further processing.

After the processing has been done, the first output channel carries the resulting events while the second one produces the construct. In both phases the actual channel names - first and second - don't matter, as any of these actions might as well be performed by either channel.

The actions described above are themselves process applications; they may be thought of as micro-process applications. Their role, in fact, may be likened to that of microprograms in computer architecture.

Henceforth the following sequence of operations defines a process application P:

(6.1)

occur $E = \{e_1, e_2, ..., e_n\} \subset C$

get $p : p \in C$

apply $p : (E, c) \rightarrow (E', c')$

occur $E' = \{e_1', e_2', ..., e_m'\} \subset C$

The sequence (6.1) is assumed to be indivisible in the sense that it always transforms initial events and constructs into some resulting events and constructs, irrespective of whether or not the process application terminates successfully.

We say that a process application **terminates successfully** (succeeds) iff:

1. The original construct $c \in C$ got processed and the result c' is consistent with the rules of syntax and semantics, **and**

2. The resulting events $E' \in C$ and trigger at least one further process application

Otherwise the application **terminates unsuccessfully** (fails), in which case we require that the original construct remains unchanged, i.e. c'=c but $E' \neq E$.

While the 'all or nothing' approach to the constructs being processed is a safeguard against potential inconsistencies, the full knowledge, i.e. that an unsuccessful attempt to perform a process was made and that attempt failed due to certain known reasons, is incorporated in E'.

A process application

(6.2) $I : (E, c) \rightarrow (E, c)$

is called the **identity process application**. It has a distinctive property of not being observable, i.e. the very fact that it has occurred cannot be detected.

A process application S that holds a construct for a period of time (e.g. to synchronise the actions of other process applications) is defined as follows:

(6.3) $S : (E, c) \rightarrow (E', c)$

where $E' = \lambda t.E(t+const)$

is potentially of great importance as it unifies in a sense two traditionally different notions - the concept of processing as different from the concept of store (in particular, the often-used word 'memory' attracts static associations).

It is worth mentioning that both input and output channels are objects of the Real World, and so are the acts of connecting and disconnecting them. As such, they are of course represented by the appropriate constructs with all the necessities of syntactic and semantic definitions.

Since, in principle, any construct may be subjected to some process application, so may a construct representing any of the abovementioned connections. This facility provides the means to describe a 'self-modifying' system whose subsequent actions are governed by the results of the previous ones.

# 7 RELATIONS OF PRECEDENCE FOR PROCESS APPLICATIONS

In order to be able to express the mutual temporal relationships between any two process applications we need to define basic operators that relate these applications in terms of precedence. These operators together with their properties (such as associativity, commutativity, or distributivity) will form the basis for a process calculus.

The first issue to be considered is **the axiom of extensionality**, regarded as a definition of equivalence of two process applications. We rephrase its standard formulation [23, 88] (as it applies to functions and sets) so as to be applicable here:

Whenever two process applications

$$P_1 : (E_1, c_1) \rightarrow (E'_1, c'_1)$$

$$P_2 : (E_2, c_2) \rightarrow (E'_2, c'_2)$$

having been triggered by the same events produce identical resulting constructs and identical resulting events for all possible arguments drawn from the set of constructs the appropriate process declarations are defined on, they are said to be **strictly equivalent.**

(7.1) $P_1$ eqv $P_2$ iff ($E_1 = E_2$ and $c_1 = c_2$) implies ($E'_1 = E'_2$ and $c'_1 = c'_2$)

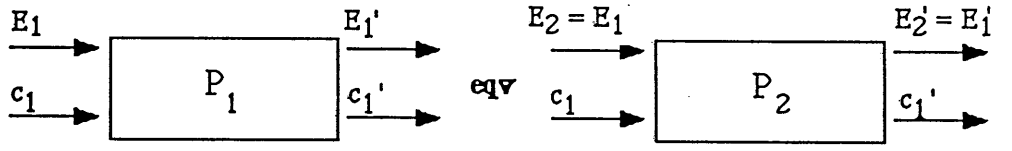



Fig.7.1

From the above definition it follows that strict equivalence is transitive:

(7.2) ($P_1$ eqv $P_2$) and ($P_2$ eqv $P_3$) implies ($P_1$ eqv $P_3$)

Three other kinds of equivalence (weak equivalence or approximation) can be defined. In these cases the equality restriction is limited to the corresponding constructs only.

We say that $P_1$ approximates $P_2$ subject to triggering events:

(7.3) $P_1$ 1-aprx $P_2$ iff ($E_1 = E_2$ and $c_1 = c_2$)
implies ($E'_1 \neq E'_2$ and $c'_1 = c'_2$)

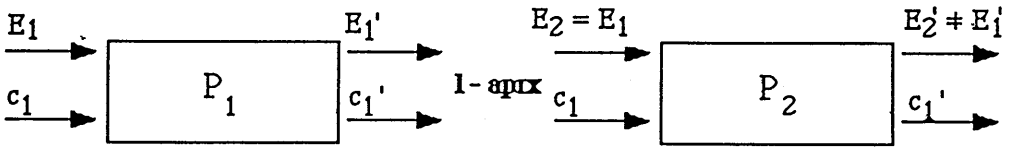



Fig. 7.2

Corespondingly, P1 approximates P2 subject to resulting events:

(7.4) $P_1$ **r-aprx** $P_2$ **iff** $(E_1 \neq E_2$ **and** $c_1 = c_2)$

**implies** $(E'_1 = E'_2$ **and** $c'_1 = c'_2)$

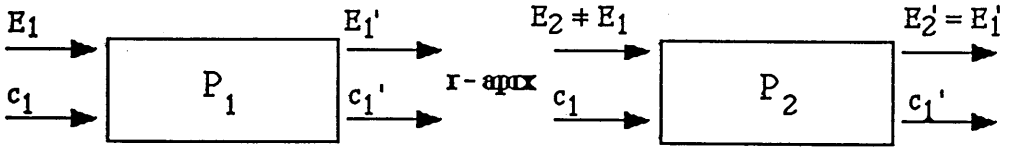



Fig. 7.3

The weakest form of approximation is expressed by the formula:

(7.5) $P_1$ **w-aprx** $P_2$ **iff** $(E_1 \neq E_2$ **and** $c_1 = c_2$

**implies** $(E'_1 \neq E'_2$ **and** $c'_1 = c'_2)$

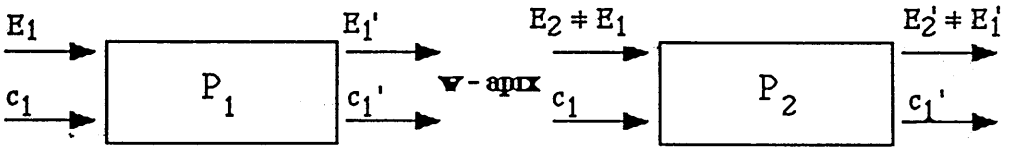



Fig.7.4

DEFINITION OF STRICTLY SEQUENTIAL PROCESS APPLICATION

Two process applications

$$P_1 : (E_1, c_1) \rightarrow (E'_1, c'_1) \text{ and } P_2: (E_2, c_2) \rightarrow (E'_2, c'_2)$$

are said to be strictly sequential ($P_1$ precedes $P_2$, $P_1$ **prec** $P_2$)

(7.6) $(P_1 >\!\text{-}\ P_2)$ **iff** **(before** $(E'_1, E_2)$ = **true)**

LEMMA: The strict sequence of process applications is transitive

(7.7) $((P_1 >\!\text{-}\ P_2)$ **and** $(P_2 >\!\text{-}\ P_3))$ **implies** $(P_1 >\!\text{-}\ P_3)$

PROOF

From (7.6) and (4.15) we have:

$(P_1 >\!\text{-}\ P_2)$ **iff** **succt(when** $(E_1')) \leq_t$ **predt(when**$(E_2))$

$(P_2 >\!\text{-}\ P_3)$ **iff** **succt(when** $(E_2')) \leq_t$ **predt(when**$(E_3))$

The functions **succt** and **predt** map to a specific time interval (5.8, 5.10), hence

$(P_1 >\!\text{-}\ P_2)$ **iff** $t'_1 \leq_t t_2$ and

$(P_2 >\!\text{-}\ P_3)$ **iff** $t'_2 \leq_t t_3$

Since $t'_2 \leq_t t'_2$ and the relation $\leq_t$ is transitive (5.1a)

$$t'_1 \leq_t t_3$$

PROOF_END

DEFINITION OF THE STRICT INVERSE SEQUENCE

Two process applications

$$P_1: (E_1, c_1) \rightarrow (E'_1, c'_1) \text{ and } P_2 : (E_2, c_2) \rightarrow (E'_2, c'_2)$$

are said to be **strictly inversely sequential** ($P_1$ follows $P_2$ or $P_1$ **fllw** $P_2$)

(7.8) $P_1 -\!< P_2$ **iff after**$(E_1, E'_2)$ = true

The definitions (7.7) and (7.8) are completely symmetrical. Therefore we have:

(7.7a) $((P_1 -\!< P_2)$ **and** $(P_2 -\!< P_3))$ **implies** $(P_1 -\!< P_3)$

(7.9) $(P_1 >\!- P_2)$ iff $(P_2 -\!< P_1)$

DEFINITION OF STRICT PARALLELISM

Two process applications

$$P_1: (E_1, c_1) \rightarrow (E'_1, c'_1) \text{ and } P_2 : (E_2, c_2) \rightarrow (E'_2, c'_2)$$

are said to be strictly parallel ($P_1 \,||\, P_2$ or $P_1$ **prls** $P_2$)

(7.10) $(P_1 || P_2)$ **iff** (**during**$(E_1,E_2)$ = true **and** **during**$(E'_1,E'_2)$ = true)

This definition means that both $P_1$ and $P_2$ must be done at the same time; otherwise $(P_1 || P_2)$ is inexecutable i.e. $(P_1 || P_2)$ fails.

LEMMA: Strict parallelism is transitive

(7.11) $((P_1 \| P_2)$ **and** $(P_2 \| P_3))$ **implies** $P_1 \| P_3)$

PROOF

From (7.11) and (4.17)

$(P_1 \| P_2)$ **iff** $att(when(E_1) = att(when(E_2))$ **and**

$(att(when(E'_1)) = att(when(E'_2))$

and

$(P_2 \| P_3)$ **iff** $att(when(E_2) = att(when(E_3))$**and**

$(att(when(E'_2)) = att(when(E'_3))$

Using the formula (4.12) we have three possible alternatives, two of them - **bottom** and **top** - are trivial (i.e.**bottom** for applications that never occurred and **top** for applications lasting forever). The remaining case will give:

$$(P_1 \| P_2) \text{ iff } t_1 = t_2 \text{ and } t'_1 = t'_2$$

$$(P_2 \| P_3) \text{ iff } t_2 = t_3 \text{ and } t'_2 = t'_3$$

and therefore $t_1 = t_3$ and $t'_1 = t'_3$

PROOF_END

It follows from the definition that strict parallelism is symmetric

(7.12) $(P_1 \| P_2)$ **eqv** $(P_2 \| P_1)$

While in many cases some process applications will have to be performed in parallel, the actual period of time needed to perform these may be different - the process applications need to be synchronised.

For that reason, three weaker forms of strict parallelism have been introduced:

(7.13) $(P_1 \mid: P_2)$ **iff** $\mathbf{during}(E_1,E_2) = \text{true}$

(i.e. P1 and P2 must start together)

(7.14) $(P_1 :\mid P_2)$ **iff** $\mathbf{during}(E'_1,E'_2) = \text{true}$

(i.e. P1 and P2 must end at the same time)

(7.15) $(P_1 >\!< P_2)$ **iff** $\mathbf{predt}(\mathbf{when}(E_2)) \leq_t \mathbf{predt}(\mathbf{when}(E_1))$ **and**
$\mathbf{predt}(\mathbf{when}(E'_1)) \leq_t \mathbf{predt}(\mathbf{when}(E'_2))$

(i.e. $P_1$ must start later and finish earlier then $P_2$)

It is perhaps easier to investigate these forms of parallelism by introducing the notion of the partial sequence.

**DEFINITION OF PARTIAL SEQUENCE**

Two process applications

$$P_1: (E_1, c_1) \rightarrow (E'_1, c'_1) \text{ and } P_2 : (E_2, c_2) \rightarrow (E'_2, c'_2)$$

are said to be **partially sequential** (P1 >÷ P2 or P1 sync P2)

(7.16) $(P_1 >\div P_2)$ **iff** $\mathbf{before}(E_1,E_2) = \text{true}$ **and** $\mathbf{before}(E_2,E'_1) = \text{true}$

**and** $\mathbf{succt}(\mathbf{when}(E_1')) \leq_t \mathbf{predt}(\mathbf{when}(E_2))$

The definition (7.16) express the fact that $P_2$ must commence before $P_1$ terminates; on the other hand it's too late to start $P_2$ after $P_1$ terminates.

Partial sequence may be considered as a most general form of a precedence-type relation, from which all of $\{>\text{-}, \text{-}<, \|, \iint\}$ may be deduced.

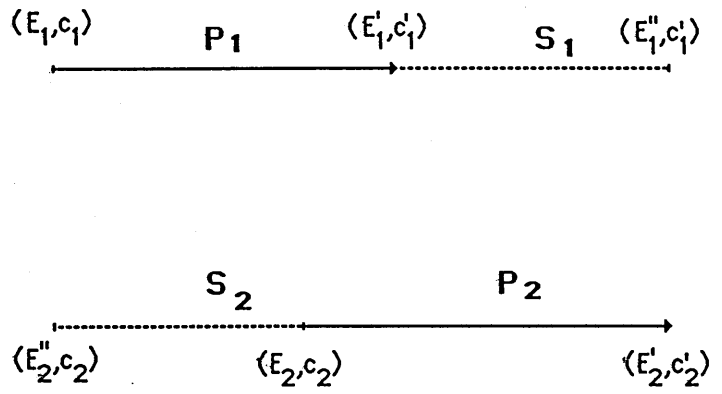



Fig. 7.5

This needs two synchronizing applications (synchronizers)

$$S_1 : (E_1, c_1) \rightarrow (E''_1, c'_1)$$

$$S_2 : (E_2'', c_2) \rightarrow (E_2, c_2)$$

where $E''_1 = \lambda t.\ E'_1(t)$ and $E'_2 = \lambda t.\ E''_2(t)$

Thus we have

(7.17) $P_1 \succ\div P_2$ eqv $(P_1 \succ\!\!- S_1) \,\|\, (S_2 \succ\!\!- P_2)$

For $S_2 = I$ we obtain

(7.13a) $P_1 \,|\!:\, P_2$ eqv $(P_1 \succ\!\!- S_1) \,\|\, P_2$

and for $S_1 = I$

(7.14a) $P_1 \,:\!|\, P_2$ eqv $P_1 \,\|\, (S_2 \succ\!\!- P_2)$

Similarly, for two synchronizers

$R_1 : (E_o, c_1) \rightarrow (E_1, c_1)$ and

$R_2 : (E'_1, c_1) \rightarrow (E''_1, c'_1)$

$R_1 \succ\!\!- P_1 \succ\!\!- R_2$

the expression (7.15) gets transformed to

(7.15a) $P_1 \succ\!\!< P_2$ eqv $(R_1 \succ\!\!- P_1 \succ\!\!- R_2) \,\|\, P_2$

Note, that for $\mathbf{att(when(E'_1))} \leq_t \mathbf{att(when(E_2)}$

$(P_1 \succ\div P_2)$ eqv $(P_1 \succ\!\!- P_2)$

while for

$\mathbf{att(when(E_1))} = \mathbf{att(when(E_2)}$ and $\mathbf{att(when(E'_1))} = \mathbf{att(when(E'_2)}$

$(P_1 \succ\div P_2)$ eqv $(P_1 \,\|\, P_2)$

DEFINITION OF FREE PARALLELISM

Two process applications

$$P_1 : (E_1, c_1) \rightarrow (E'_1, c'_1) \text{ and } P_2 : (E_2, c_2) \rightarrow (E'_2, c'_2)$$

are said to be **freely parallel** ($P_1 \iint P_2$ or $P_1$ **prrl** $P_2$) if their execution can be done in either order of precedence or concurrently:

(7.18) $P_1 \iint P_2$ **iff** ( $P_1 \| P_2$ **or** $P_1 >\!\!- P_2$ **or** $P_1 -\!\!< P_2$)

LEMMA: Free parallelism is transitive.

(7.19) (($P_1 \iint P_2$) **and** ($P_2 \iint P_3$)) **implies** ($P_1 \iint P_3$)

PROOF

From definition (7.18) we have:

$P \iint Q$ **iff** $P >\!\!- Q$ **or** $Q >\!\!- P$ **or** $P \| Q$

$Q \iint R$ **iff** $Q >\!\!- R$ **or** $R >\!\!- Q$ **or** $Q \| R$

$P \iint R$ **iff** $P >\!\!- R$ **or** $R >\!\!- P$ **or** $P \| R$

If transitivity is to hold then the following formula (derived from 7.19 by removing implication and then applying de Morgan's Law) has to be a tautology:

(7.19a) **not**($P >\!\!- Q$ **or** $Q >\!\!- P$ **or** $P \| Q$) **or**

**not**( $Q >\!\!- R$ **or** $R >\!\!- Q$ **or** $Q \| R$) **or**

($P >\!\!- R$ **or** $R >\!\!- P$ **or** $P \| R$) = true

We prove that (7.19a) is indeed a tautology by considering all possible cases, bearing in mind that the logical value of disjunction is false only when all components of that disjunction are false.

The following cases:

$$P >\!- Q \text{ and } Q >\!- R \quad \text{and} \quad Q >\!- P \text{ and } R >\!- Q$$

$$P \| Q \text{ and } Q \| R$$

$$P >\!- Q \text{ and } Q \| R \quad \text{and} \quad Q >\!- P \text{ and } R \| Q$$

$$P \| Q \text{ and } Q >\!- R \quad \text{and} \quad P \| Q \text{ and } R >\!- Q$$

are all trivial since the relationships between P, Q and R determine the relationship between P and R; corespondingly (7.19) gets reduced to either transitivity of precedence (incidence) or to transitivity of strict parallelism.

The two remaining cases:

$P >\!- Q$ and $R >\!- Q$ and anti-symmetrically $Q >\!- P$ and $Q >\!- R$

need further investigation as in neither case the temporal relationship between P and R may be inferred. If $P>\!-Q$ = true and $R>\!-Q$ =true then (7.19a) may be transformed to the form:

$$\text{false or false or } (P >\!- R \text{ or } R>\!-P \text{ or } P\|R) = \text{true}$$

Since $P>\!-Q$ and $R>\!-Q$ are both true, the following holds:

$$P >\!- R \text{ or } R>\!-P \text{ or } P\|R \text{ or } P \perp R = \text{true}$$

By definition (7.21) the above formula gets transformed to:

$$P >\!- R \text{ or } R>\!-P \text{ or } P\|R \text{ or } (P >\!- R \text{ or } R>\!-P \text{ and not} P\|R)$$

$$=(P\|R \text{ or } (P >\!- R \text{ or } R>\!-P)) \text{ or } (\text{not } P\|R \text{ and } (P >\!- R \text{ or } R>\!-P))=$$

$$P\|R \text{ or } ((P >\!- R \text{ or } R>\!-P) \text{ or } ((P >\!- R \text{ or } R>\!-P) \text{ and not } P\|R))=$$

$$P\|R \text{ or } P >\!- R \text{ or } R>\!-P$$

which concludes the proof.

PROOF_END

From definition and formulae (7.9) and (7.12)

(7.20) $(P_1 \iint P_2)$ **iff** $(P_2 \iint P_1)$

**DEFINITION OF MUTUAL EXCLUSIVENESS**

Two process applications

$$P_1: (E_1, c_1) \rightarrow (E'_1, c'_1) \text{ and } P_2 : (E_2, c_2) \rightarrow (E'_2, c'_2)$$

are said to be **mutually exclusive** ($P_1 \perp P_2$ or $P_1$ **clsh** $P_2$) if their execution can be done in either order of precedence but not concurrently:

(7.21) $P_1 \perp P_2$ **iff** ( $P_1 >\!\!-\ P_2$ **or** $P_1 -\!\!< P_2$) **and not**($P_1 \parallel P_2$)

It follows from (7.9) and (7.12) that mutual exclusiveness is symmetric

(7.22) $(P_1 \perp P_2)$ **iff** $(P_2 \perp P_1)$

**LEMMA:** Mutual exclusiveness is not transitive

(7.25) $((P_1 \perp P_2)$ **and** $(P_2 \perp P_3))$ **not_implies** $(P_1 \perp P_3)$

**PROOF** [*Reductio ad absurdum*]

From definition (7.21) we have

$P \perp Q$ **iff** (( $P >\!\!-\ Q$ **or** $Q >\!\!-\ P$)) **and not**($P \parallel Q$)

$Q \perp R$ **iff** (( $Q >\!\!-\ R$ **or** $R >\!\!-\ P$)) **and not**($Q \parallel R$)

P ⊥ R **iff** (( P >- R **or** R >- P)) **and not**(P || R)

If transitivity were to hold, we would deduce a contradiction as follows. The formula

((( P>- Q **or** Q>-P)) **and not**(P||Q)) **and** (((Q >-R **or** R>-P)) **and**

**not**(Q || R)) **implies** (( P>- R **or** R >-P)) **and not**(P || R)

would have to be a tautology.

Suppose P >- Q is true; we therefore infer that Q >- P is false and so is P||Q. Similarly, if R >- Q is true, then both Q >- R and Q||R are false. Hence, the above formula may be reduced to:

(((true **or** false) **and** true) **and** ((false **or** true) **and** true)

**implies** (( P >- R **or** R >- P)) **and not**(P || R)

and consequently to

true **implies** (( P >- R **or** R >- P)) **and not**(P || R)

From the assumptions P>-Q = true and R>-Q = true no temporal relationship between P and R may be deducted. Thus for P||R = true, which means that both P >- R and R >- P are false, we obtain

true **implies** ((false or false) **and** false)

that is: true **implies** false, which is contradictory.

PROOF_END

# 8 PROCESS CALCULUS

In the previous chapters various kinds of precedence relations were investigated. The possible relationships between two process applications were identified, named and formally defined on the basis of the time model introduced in Chapter 5. Some important properties, such as reflexivity, symmetry and transitivity were also considered in detail.

The outcome of those considerations is a notation whereby an expression describing (possibly complex) temporal relationships occurring amongst a number of process applications may be formed.

We aim to turn this notation into a calculus - that is a formal system composed of expressions and operations on them. The precise definition of what is meant by an 'expression' will be given, and conversion rules will be developed. These rules will allow the handling of large expressions - to reduce them to simpler forms whenever possible, and to test them against any inconsistency that might occur.

(8.1) process_calculus= <{expression}, {logical_operator}, {precedence_rel}>

where:

logical_operator ::= **and** | **or** | **not** /*predicate calculus*/

precedence_rel::= **prec** | **fllw** | **prls** | **prrl** | **sync** | **clsh**

expression ::= term | expression logical_operator term

term ::= proc_app_var | term precedence_rel proc_app_var

proc_app_var::= *process application name* | (expression)

The above syntax generates regular expressions. Two kinds are particularly interesting:

$$(P \begin{matrix}\textbf{and}\\ \textbf{or}\end{matrix} Q)\left\{\begin{matrix}>-\\ \|\\ \perp\end{matrix}\right\} R$$

and

$$(P\left\{\begin{matrix}>-\\ \|\\ \perp\end{matrix}\right\} Q) \begin{matrix}\textbf{and}\\ \textbf{or}\end{matrix} (P\left\{\begin{matrix}>-\\ \|\\ \perp\end{matrix}\right\} Q)$$

*First*, some convention concerning priorities of precedence relations and logical operators needs to be stated. We assume that all of {>-, -<, ||, ∫∫, ⊥} are equal in priority but any precedence relation associates stronger than any logical operator; the usual convention for the latter (i.e. **not** over **and** over **or**) still applies.

Thus, for instance

$$P >\!- Q \text{ and not } R \parallel L \text{ or } Q \perp S$$

means

$$((P >\!- Q) \text{ and } (\text{not}(R \parallel L))) \text{ or } (Q \perp S)$$

rather than for example

$$(P >\!- (Q \text{ and not} R)) \parallel (L \text{ or } Q) \perp S$$

The assumption concerning equal priorities of precedence relations causes the need for brackets to avoid ambiguity for, unlike in the predicate calculus simple rules of associativity are not available. Hence for instance:

$$(P \parallel Q) >\!- R \neq P \parallel (Q >\!- R)$$

$$(P >\!- Q) \parallel R \neq P >\!- (Q \parallel R)$$

$$(P \parallel Q) \perp R \neq P \parallel (Q \perp R)$$

and so on.

The meaning of all possible cases is given below together with corresponding graphical interpretation, with the time increasing to the right:

**a)**

$P>-(Q>-R)$ $\quad \underline{P} \quad \underline{Q} \quad \underline{R} \quad (P>-Q)>-R$

$P>-Q>-R$

**b)**

$(P >\!\!- Q) -\!\!< R$ R P Q

$P >\!\!- (Q -\!\!< R)$ P R Q

**c)**

$(P -\!\!< Q) >\!\!- R$ Q P R

$P -\!\!< (Q >\!\!- R)$ Q R P

**d)**

$(P \,||\, Q) \,||\, R$

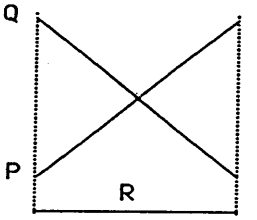
P
Q
R

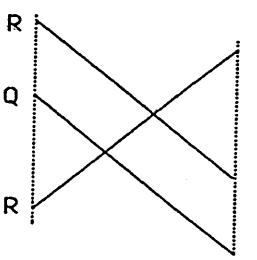
$P \,||\, Q \,||\, R$

$P \,||\, (Q \,||\, R)$

**e)**

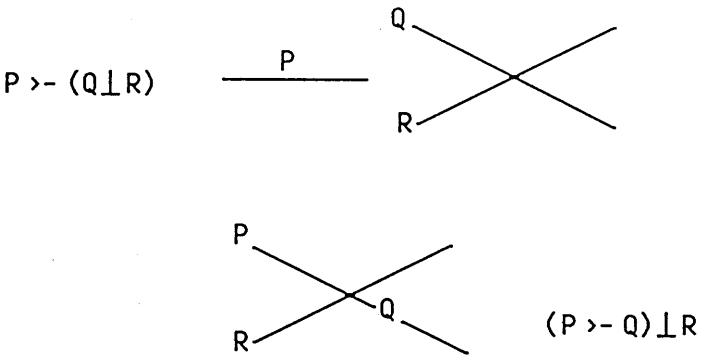
$(P \perp Q) \,||\, R$




$P \perp (Q \,||\, R)$

f)




*Second*, we take as axioms the following formulae:

(8.2) (P **or** Q) >- R = (P >- R) **or** (Q >- R)

(P **and** Q) >- R = (P >- R) **and** (Q >- R)

(8.3) (P **or** Q) ⊥ R = (P ⊥ R) **or** (Q ⊥ R)

(P **and** Q) ⊥ R = (P ⊥ R) **and** (Q ⊥ R)

(8.4) (P **or** Q) ∥ R = (P ∥ R) **or** (Q ∥ R)

(8.5) (P **and** Q) ∥ R = (P ∥ R) **and** (Q ∥ R)

Since the strict parallelism is both symmetric and transitive, the formula (8.5) may be transformed to

(8.6) (P **and** Q) ∥ R **implies** (P ∥ Q)

LEMMA: The following formulae are valid.

(8.7) $$(P \iint Q) \text{ or } \left(P \left\{ \begin{matrix} >- \\ \| \\ \perp \end{matrix} \right\} Q\right) \text{ iff } (P \iint Q)$$

PROOF

Whenever substitutions are made, the following denotations hold: $P >- Q = p$,

$Q >- P = q$, $P \| Q = x$, $P \perp Q = y$, p or q = t, so $P \iint Q = x$ or (p or q)).

Case (a)

$(P \iint Q \text{ or } P >- Q) = x$ or p or q or p = x or p or q = $P \iint Q$

Case (b)

$(P \iint Q \text{ or } P \| Q) = x$ or p or q or x = x or p or q = $P \iint Q$

Case (c)

$(P \iint Q \text{ or } P \perp Q) = (x$ or p or q) or ((p or q) and not x) =

(x or t) or (not x and t) = x or (t or (t and not x)) =

x or t = x or p or q = $P \iint Q$

PROOF_END

LEMMA: The following formulae are valid.

(8.8) $$(P \iint Q) \text{ or } \left(P \left\{ \begin{matrix} \gtrdot \\ \| \\ \perp \end{matrix} \right\} Q\right) \text{ iff } \left(P \left\{ \begin{matrix} \gtrdot \\ \| \\ \perp \end{matrix} \right\} Q\right)$$

PROOF

As previously, the following denotations hold: $P \gtrdot Q = p$, $Q \gtrdot P = q$, $P \| Q = x$, $P \perp Q = y$, p or q = t, so $P \iint Q = x$ or (p or q)).

Case (a)

$(P \iint Q$ and $P \gtrdot Q) = (x$ or p or q) and p =

p and (p or (x or q)) = p = $P \gtrdot Q$

Case (b)

$(P \iint Q$ and $P \| Q) = (x$ or p or q) and x =

x and (x or (p or q) = x and (x or t) = t = $P \| Q$

Case (c)

$(P \iint Q$ and $P \perp Q) = (x$ or p or q) and ((p or q) and not x) =

(x or t) and (t and not x) = (x or t) and t) and not x =

t and not x = (p or q) and not x = $P \perp Q$

PROOF_END

The central theorem in the process calculus here concerns its **precedential completeness** The concept of precedential completeness is based on the notion of completeness discussed in [19,40,91]. Here, this term denotes the fact that every expression which can be formed in the notation of the calculus and is true in terms of precedence relations, is a consequence of axioms these precedence relations are devised from. We aim to prove that the calculus is complete by finding an invariant **Inv**(P,Q) that is always true for any two process applications P and Q occurring in a time-space T.

LEMMA:

(8.9) Inv(P, Q) = **not** ((P>-Q) **or** (Q>-P) **and not**(P∫∫Q)) = true

PROOF

**not** ((P>-Q) **or** (Q>-P) **and not**(P∫∫Q)) =

**not** ((P>-Q) **or** (Q>-P) **and not** ((P>-Q) **or** (Q-< P) **or** (P||Q)) =

**not** ((P>-Q) **or** (Q>-P) **and** (**not** ((P>-Q) **or** (Q-< P))) **and not**(P||Q)) =

**not** ( false **and not**(P||Q) = true

PROOF_END

CONCLUSION: The set {>-, ∫∫} is precedentially complete and so is {>-, ||}

LEMMA:

(8.10) Inv(P, Q) = **not** ((P||Q) **and** (Q ⊥ P)) = true

PROOF

**not** ((P||Q) **and** (Q ⊥ P)) = not (P||Q **and** (P>-Q **or** Q>-P) **and not**(P||Q) = true

PROOF_END

CONCLUSION: The set {||, ⊥ } is precedentially complete.

LEMMA:

(8.11) Inv(P, Q) = not (not (P>-Q or Q>-P) and P ⊥ Q )= true

PROOF

not (P>-Q or Q>-P) and P ⊥ Q =

not (P>-Q or Q>-P) and ((P>-Q or Q>-P) and not P||Q) =

not (P>-Q or Q>-P) and (P>-Q or Q>-P) and not P||Q = false

PROOF_END

CONCLUSION: The set {>-, ⊥ } is precedentially complete.

We conclude these investigations by formulating **completeness theorem**, which is in fact a corollary stemming from the above lemmas.

COMPLETENESS THEOREM

**Any two operators from the set {>-, ||, ⊥ } form a precedentially complete set**

This theorem is of a particular importance, as it shows that whenever a precedence relationship between two process applications is to be defined, at most two of the three basic temporal relations (i.e. strict sequence, strict parallelism or mutual exclusiveness) will suffice.

This theorem will also allow to infer the

PRINCIPLE OF STRUCTURED PROCESS DESIGN:

A process application P can be decomposed into two process (sub)applications $P_1$ and $P_2$ whose mutual temporal relationship is expressed by one of the basic temporal relation:

$$P = P_1 >- P_2 \quad \text{or} \quad P_1 \| P_2 \quad \text{or} \quad P_1 \perp P_2$$

From the axiom of extensionality it follows that decomposition of a process application can be done in many different ways. An interesting problem is to determine, given syntax and semantics of all the constructs of a particular system, how to form an *optimal decomposition*. This potentially important research problem needs however further investigations - it requires, in the first place, the optimization criteria to be formulated.

# 9 THE IMPACT OF PROCESS STRUCTURE ON DATA MODEL

This chapter discusses the consequence of not making the data model immune from the process structure. This follows the thesis expressed in Chapters 1 and 2 that data model should not be designed in isolation from the processes that are to operate on that data, especially if these processes are not sequential.

The principle of central control of corporate information laid down the foundations for database theory. The database characteristic features of data shared amongst various applications, controlled redundancy and data independence are all derived from this principle. Inevitably the development of database theory has centred around the issues of store and processing economy. An optimal structure of sets of data has been sought.

In the relational approach the initial structure is a set of tuples, each tuple consisting of attributes. Any two tuples from this set must have the same structure, that is must contain exactly the same kinds of attributes. Attributes within the set depend somehow on each other; these relationships are the result of the semantics of the data. They are called functional, multivalued and join dependencies.

Some structures attract certain undesirable properties - the so-called insertion, deletion and update anomalies. The problem is then to find a structure that is free from these anomalies, thus ensuring consistency of data in the database - consistency threatened by some operational actions. The optimal structure must be capable of carrying exactly the same information contents as the initial one. The optimization criterion is that any possible increase of data volume due to structure transformation is to be minimised.

A general solution that was found to that problem is called normalization. In the relational theory it is a discrete finite algorithm producing a family of relations - all derived from that initial set. The mechanism of transformation from one structure to another is based on functional, multivalued and join dependencies; it ensures that the final relations are free from the anomalies.

No attention is paid to the efficiency of processing, or indeed to processing as such. It has been assumed that an appropriate Data Sub-Language (DSL) should guarantee executability of any process, be it retrieval, updating or restructuring. Moreover, none of these processes is necessarily considered in appropriate detail when the data model is being designed. The theory also assumes (as a consequence of the 'sequentiality' principle) that whenever consistency constraints are being threatened, the processes performing updating or restructuring must have greater priority than (or finish before) any interrogation.

In general - though the traditional trade-off *'storage structure versus process complexity'* was resolved by ignoring the latter - normalization theory yields reasonably correct data models, provided that the following four conditions are satisfied:

1. **Updating is simple**

   Any change that may potentially occur to a database can be reflected in a single expression in DSL (that is an expression composed of a single insert/delete statement). It is, moreover considered that modifying a data element in one relation will not necessitate any change in another. More precisely this problem is usually left to an external intervention - mostly human though sometimes with the use of a Data Dictionary Facility.

– However, in many applications the logic of updating is often complex and may require a number of relations to be updated [86]. The usual remedy is to design updating processes as indivisible transactions. Each transaction supports all the necessary actions and for its whole duration no other access to the relations concerned is permitted.

   This solution, however, cannot be appreciated from a theoretical viewpoint. Transaction design is not normally done at the conceptual level and, indeed, is distinct from data modelling. In effect, large portions of databases are too often locked denying access to interrogations which, by and large, are main functions of any information system.

2. **Restructuring is non-frequent**

   Each data model is believed by its designer and users to be stable, i.e. independent to some extent over time. Therefore the period in between subsequent restructurings (however complex they might be) is considered long enough for operational purposes. During these periods the database is then in a time-independent state. While this assumption may be taken as appropriate for certain applications, it cannot be accepted for some others. A road database that contains geometrical and technical information is an example of a fairly stable system. The road network (i.e. a system of connections that is a reference base for other information) does not change

very often, nor does a need for a new attribute appear too frequently. The usual period between subsequent restructurings is long enough (say a year) to accept the data model's stability.

A traffic control system is, however, a different case. It is a quasi real-time system where both the values of attributes and the attributes themselves may have to be modified to reflect the actual traffic conditions.

- In general, restructuring is seen as a sort of 'house-keeping' process (and concerned with physical rather than logical aspects of a database) whose impact on the conceptual data model may be ignored.

**3. The size of data is manageable**

This condition express a belief that the database, quite irrespective of its structure is small enough for any processing to be done in a reasonably short period of time. Again, the details of such processing are rarely taken into account at the phase of conceptual data modelling.

In most more advanced applications this condition will sooner or later be violated. Files will inevitably grow in size (particularly in those databases with so-called historical data) and get restructured many times over, until their inertia will necessitate a total reconstruction.

There is no simple solution to this problem. This issue is often referred to as 'database tuning' - an activity based on performance statistics and practical experience rather than on some formal theory that is applicable at the level of conceptual design.

4. **Data model is independent from processing**

   Behind this condition there is an implicit assumption that all processes (at least conceptually) are performed sequentially. That is, if two processes require access to a particular relation one of them must wait.

   At the conceptual level no processing is considered as being influential enough to revise the data model. However, designers (perhaps unconsciously) realise the possible influences of, say, a particular type of a query that may frequently be asked by the database users. They apply various *programming* techniques (e.g. secondary indexing, relation ordering) to repair in a practical way faults in the data model; the faults which could have been avoided if the designers were able to use theoretically sound means at the level of conceptual modelling.

The objective of this chapter is to investigate the effects which may occur if the hypothesis that the data model is immune from process structure is rejected. I aim to achieve this objective by considering in some detail the impact of three basic precedence relations (**prec, prls, clsh**) occurring between two process applications on the constructs these applications act upon. The concept of access and store anomalies (quite analogously to update anomalies) is introduced. Later in this chapter the notion of two kinds of process dependence will also be introduced to explain some possible transformations of a relational data model to the more suitable (but still relational) form that reflects better the relevant process structure.

**Access anomaly** refers to a situation where a process application P is denied an access to a (large) set of construct C due to another action, i.e. process application Q which, having a higher priority than P, occurs at the same time. P requires an access to $C_1$ while Q performs an action on $C_2$; both $C_1$ and $C_2$ are subsets of C, but $C_1$ is disjoint with $C_2$.

**Store anomaly** describes a situation where two process applications P and Q temporally connected by any of {**prec, prls, clsh**} require an access to one and the same construct **c** in its original form. Both P and Q, whatever their precedence relationship, process **c** producing $c_p$ and $c_q$ respectively.

The access anomaly is considered as an undesirable property of a particular data structure since it causes unnecessary delays in processing.The store anomaly is also considered undesirable as it may lead to an uneconomical use of store.

In the previous chapters the relations of precedence and their properties, and the process calculus were all devised and considered purely in terms of temporal relationships. In fact, no reference was made to the actual constructs the relevant process applications were supposed to act upon. The only assumption was that the appropriate constructs (i.e. events, process declarations and constructs to be processed) were available whenever conducive circumstances made processors to take necessary actions.

We now consider two process applications:

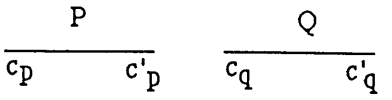
$$P: (E_p,c_p) \rightarrow (E'_p,c'_p)$$

$$Q: (E_q,c_q) \rightarrow (E'_q,c'_q)$$

assuming that either $c'_p = c_q$ or $c_p = c_q$ may occur whatever temporal relationship between P and Q holds.

Without loss of generality (due to the completeness theorem) the following three cases are considered: P **prec** Q, P **prls** Q and P **clsh** Q.

**CASE 1:** P **prec** Q




a) $c'_p = c_q$

The result of P is taken by Q as input.

A traditional pattern of sequential processing. No anomaly occurs and no re-design is needed.

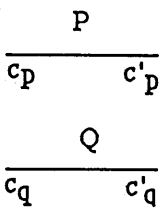
b) $c_p = c_q$

Both P and Q are to process the same construct producing perhaps different results.

Store anomaly occurs so re-design is needed. Two possible solutions appear:

1. Multiple copies of $c_p$, which may prove uneconomic if a large number of process applications are involved. Also, this is contradictory to the principle of data consistency.
2. All process declarations that are to be applied on $c_p$ must contain a sub-construct (either pre-condition or post-condition) to restore the original form of $c_p$ before passing it to the process applications that follow.

**CASE 2:** P **prls** Q




a) $c'_p = c_q$

Contradiction

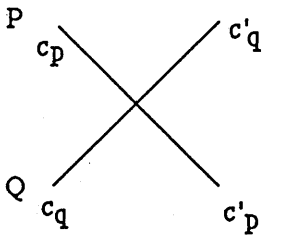
b) $c_p = c_q$

If $c_p$ is an atom then store anomaly occurs. So, if parallelism is to be retained then multiple copies of $c_p$ must be kept. Otherwise (i.e. if for some reason the designer does not want multiple copies) the process application must be executed in either of the forms:

P **prec** Q or P **fllw** Q or P **clsh** Q.

Non atomic constructs may be decomposed. If $c_p$ can be decomposed into $c_{pp}$ and $c_{pq}$ such that P operates exclusively on $c_{pp}$ and Q operates exclusively on $c_{pq}$, and both $c_{pp}$ and $c_{pq}$ are meaningful, then access anomaly for $c_p$ occurs and re-design is needed.
If such decomposition is impossible then, as previously, P **prls** Q is inexecutable.

**CASE 3:** P **clsh** Q




a) $c'_p = c_q$

By definition (7.21)

(P **clsh** Q) = (P **prec** Q) **or** (Q **prec** P) **and not** (P **prls** Q)

A strictly non-deterministic approach requires that at any instant of time *either* of the two sequences P **prec** Q and Q **prec** P would have to be executable. This however leads to a contradiction since the sequence Q **prec** P, i.e.

$(c'_p \rightarrow c'_q) >\!\!- (c_p \rightarrow c'_p)$ is a nonsense.

Taking a deterministic view, the application (P **clsh** Q) would be restricted to (P **prec** Q) which corresponds to the Case 1a.

b) $c_p = c_q$

This is equivalent to Case 1b with two possible sequences:

$$(c_p \rightarrow c'_p) >\!\!- (c_p \rightarrow c'_q)$$

$$(c_p \rightarrow c'_q) >\!\!- (c_p \rightarrow c'_p)$$

In both cases store anomaly occurs.

In the relational data model, as mentioned earlier, normalization - a technique based on the notion of functional dependence - ensures that the final family of relations is free from undesirable properties, that is from update anomalies.

We aim to show that these normalized relations may still possess some other properties, i.e. access anomalies, that are considered harmful. They cannot be removed on the grounds of normalization theory. For, though some relations may need to be decomposed, neither functional nor multivalued nor join depen- dencies provide sufficient mechanism for such decomposition. Therefore two other kinds of dependence must be defined.

Suppose we have a relation R composed of three attributes - X, A and B. We assume that R is in Boyce-Codd Normal Form (BCNF) and that X is the primary non-composite key in R. Therefore two functional dependencies $X \rightarrow A$ and $X \rightarrow B$ hold for R and no other dependence holds for R.

We say that a **tangled process dependence** holds for R iff the following conditions are satisfied:

1. **there is** process application P: $(E_p, (X,A)) \rightarrow (E'_p,(X,A'))$
2. **there is** process application Q: $(E_q, (X,B)) \rightarrow (E'_q,(X,B'))$
3. the temporal relationship between P and Q is of the form P **prec** Q or Q **prec** P or P **clsh** Q at every instant of time.

Clearly, if a tangled process dependence holds for R an access anomaly occurs whenever P and Q become operationally active. The attributes A and B are called process-independent with respect to the process applications P and Q. To remove the access anomaly it is sufficient to decompose R into two of its projections: $R_1(X,A)$ and $R_2(X,B)$; R is the equi-join of these.

As an example consider a relation from a road database (Fig. 9.1 shows a fictitious possible instance of this relation):

TRAFFIC-SPACE (SECTION_ID, TRAFFIC_DENSITY, ACCIDENT_RATIO).

From a phenomenological viewpoint of a traffic engineer this relation represents a correct and coherent model of a Real World entity that describes driving conditions on a number of road sections. SECTION_ID identifies uniquely every particular road section that carries a certain amount of traffic (TRAFFIC_DENSITY, in vehicles per day) and has certain level of danger measured by ACCIDENT_RATIO (in number of vehicles per mile per year). The last two data items are indeed very closely related in any sort of analytical job for traffic engineering though no functional dependence between these two can be found.

| SECTION_ID | TRAFFIC_DENSITY | ACCIDENT_RATIO |
|---|---|---|
| M1J1 - M1J2 | 70 000 | 2.8 |
| M1J2 - M1J3 | 75 000 | 1.8 |
| M1J3 - A1B538 | 35 000 | 2.5 |
| M25J24 - A1081B513 | 5 000 | 3.2 |

Fig. 9.1

However, from an operational point of view the above design needs to be improved, considering processes likely to be applied on this relation. Let's consider, for instance, two of them - UPDATE TRAFFIC and UPDATE ACCIDENT. In practice, the traffic density values are modified at certain intervals (say two years) that normally differ from those at which the accident data get updated. Also, these two processes are carried out separately and by different group of people. It is, therefore, unreasonable to have the whole relation locked during the execution of either updating, preventing any access to the attributes unaffected.

Hence, the more convenient model would consist of two relations:

TRAFFIC (SECTION_ID,DENSITY) and ACCIDENT (SECTION_ID, RATIO).

We next define the other kind of process dependence. Let R denote a relation composed of attributes X, Y and A. As previously, we assume that R is in BCNF and X is the primary non-composite key in R. Only one functional dependence $X \rightarrow A$ exists. The attribute Y plays the role of a secondary index so, for instance R could be inverted with respect to Y. The attribute A denotes a particular kind of, say, technical data whose nature is irrelevant to this consideration.

This relation can then be seen as an union of its restrictions:

$$R(X,Y,A) = R_1(X,[Y=y_1],A) \cup R_2(X,[Y=y_2],A) \cup \ldots \cup R_n(X,[Y=y_n],A)$$

The denotation $[Y=y_i]$ is used to record the fact that all tuples in the relation $R_i$ have the same value (a constant $y_i$) for the attribute Y.

We say that a **flat process dependence** holds for R iff:

1. **for** i = 1,2, ... , n **there is** process application

$$P_i : (E_{pi}, (X, [Y=y_i], A)) \rightarrow (E'_{pi}, (X, [Y=y_i], A'))$$

2. for any i,k = 1,2, ... ,n

either $P_i$ **prec** $P_k$ or $P_k$ **prec** $P_i$ or $P_i$ **clsh** $P_k$

at any instance of time.

If a flat process dependence holds for R then an access anomaly occurs whenever $P_i$ and $P_k$ become active. The obvious way of dealing with the access anomaly here is to split the relation R into a family of its restrictions as shown above.

An example offered here (Fig.9.2 shows a schema of a possible instance) is drawn again from a road database:

SECTION-GEOM (SECTION_ID, REGION_NO, geometrical data)

This relation contains some geometrical information (such as width, length, permissible height of vehicles) about road sections uniquely identified by SECTION_ID; each section is located within certain maintenance area identified by REGION_NO.

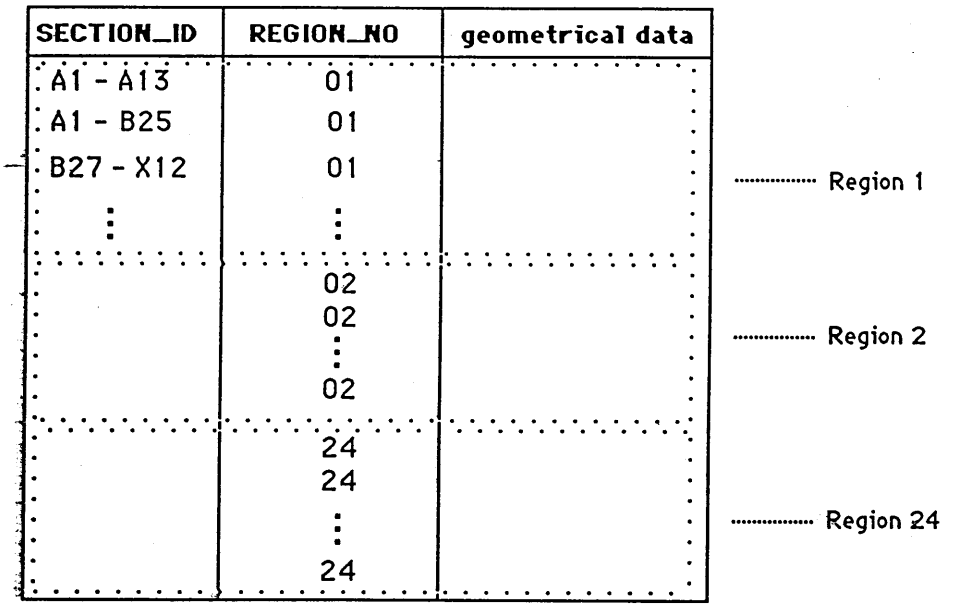



Fig. 9.2

Usually, updating the values of geometrical data lies within the responsibilities of regional road authorities. Also, not all queries involve necessarily the whole relation. Hence maintaining all data within one relation seems uneconomic, cumbersome and causing delays in processing.

The problem has strong quantitative bias as the real-life databases may (and do) comprise as many as hundreds of this kinds of relation and the volume of each may exceed tens of megabytes.

Note that tangled process dependence may be seen as a way of detecting under-abstraction while flat process dependence corresponds to a specific case of over-abstraction (see Chapter 1).

# 10 CONCLUSIONS

In Chapter 1 I discussed several reasons why the 'State-of-the-Art' of system design theories seemed to me quite unsatisfactory. These reasons of dissatisfaction arose both from some theoretical considerations and from my practical experience. For, when designing systems, I had many times found myself in situations where either no theory provided a mechanism to devise a solution to a particular problem, or for a particular devised solution no theoretical justification existed.

These were essentially the motives to undertake the present research. Its objectives were formulated with the intention to eliminate some of the existing drawbacks, particularly those that were results of concurrency constraints. Let's recall these objectives here:

- *to develop a notation that is both suitable for the description of information systems and is free from concurrency constraints*

- *to discover the rules governing decomposition (integration) of processes within a target system*

- *to investigate possible effects of process design, being performed integrally with data design, onto the data model that represents the Real World within the target system.*

I hope the results presented in this work justify the claim that the objectives were achieved. I should also like to discuss some supporting issues.

A theory whose task is to explain the act of modelling the Real World must provide a device to express any model in a way that is comprehensible by people and machine processable in the environment in which this model is to be implemented. At the same time the model must be isomorphic to the relevant part of the Real World, that is the constructs and actions occurring in the model must be in one-to-one correspondence with the objects and their behaviour in the Real World.

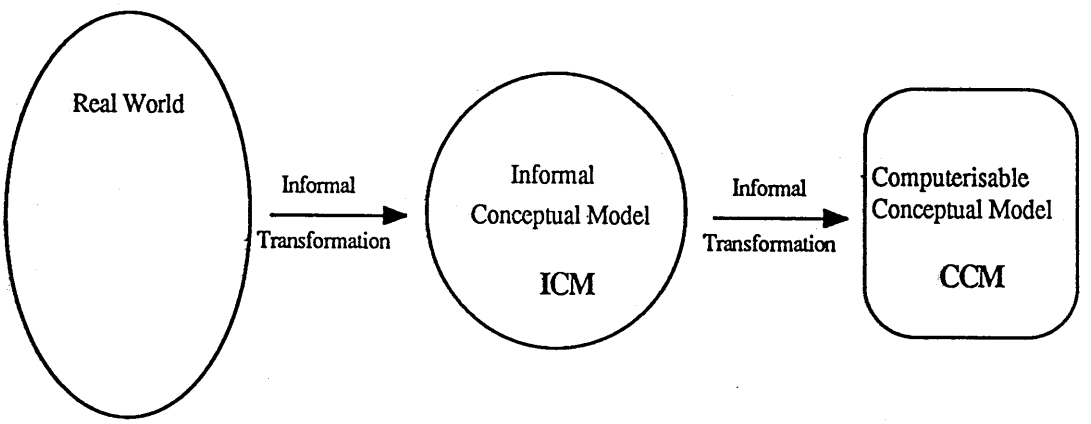



Fig. 10.1

Strictly speaking, developers create typically two models - an Informal Conceptual Model (ICM) and a Computerisable Conceptual Model (CCM) though very often they may not even realize this fact.

While transformation of the ICM (ultimately seeming coherent, consistent and complete) into the CCM may be seen as a fairly rigorous exercise, the transformation from the informal thoughts, words and actions of the Real World to a systematic form of ICM is not. This transformation, itself unspecifiable and inexact as done by humans, is disturbed even further by the influence of some questions deeply rooted

in people's minds: the questions which they are quite unable to separate themselves from while doing this transformation, despite, perhaps, their awareness that these would have to be considered either later or at least separately.

Among others, there are matters related to the choice of strategy and reasons for which the system is being created and to the choice of subset (i.e. which phenomena, objects and actions are to be included in the model). These are, of course, valid questions to be answered. But neither the relevant solutions nor the actual method to find them should affect the way the Real World is modelled. For example the von Neumann model of computing affects system analysis to the extent that actions are very often modelled as sequential irrespective of (or contrary to) their actual behaviour in the Real World.

These were, then, the conditions that suggested the fundamental hypothesis concerning the necessity of integral treatment of the three aspects of modelling - syntax, semantics and behaviour.

The sentential notation arose from this hypothesis. It is a device allowing one to describe the objects of the Real World in a simple structured language. Expressions in that language (i.e. constructs) have been made uniform in a sense that all constructs are treated in the same way irrespective of their interpretation. Hence it is not possible on the grounds of syntactic analysis alone to detect whether, for instance, a particular construct belongs to a class of processes or to a class of data. Nor can one deduce from the syntax whether a construct represents a class, i.e. named type of similar UoD objects or represent a single UoD object. Indeed, one and the same construct may be given a number of different interpretations depending on circumstances. For example at time $t_1$ an expression is to be taken as a description of a process to be carried out by a processor, while at time $t_2$ it will be a construct subjected to a processing. A customer may be a process, an entity, or a source or a sink of information flow.

The model of semantics was built as a system of correspondences between the Universe of Discourse, the model of that Universe (recorded as a system of constructs) and the Environment of Comprehension ($E_{oc}$). The notion of the Environment of Comprehension is potentially of great importance. The analyst can construct models of semantics containing more than one $E_{oc}$ to reflect the situations where the Real World (or some parts thereof) are being conceived differently by different groups of people.

Of all the concepts behavioural issues were given the most extensive consideration, forming the main part of this thesis. The central question in describing the behaviour of a system is how the occurrences of its components (i.e. widely understood processes) are related in terms of precedence. While ignoring the exact timing of these occurrences would have made the theory simpler, the precise definitions of temporal relationships require some sort of quantification. Similarly, in order to prove certain relationships between these temporal relationships a quantified time space was required.

In consequence, the time model was devised in the form of a cartesian-like time space consisting of: a set of time-intervals with a linear ordering and a metric that defines the length of time (i.e. a distance between two time intervals). In recognition of the fact that the objects in the Real World may be temporally referenced to in more than just one time system, and indeed relatively to each other, the notions of global and local time systems related to each other through the precedence preserving function were introduced. Each time system is additionally equipped with a functional device CLOCK giving at every instant of time the 'age' of that system since its establishment.

The introduction of the time model in the form of a metric space does not necessarily mean that the location of each object in time must be precise. It does however mean that, whenever necessary, the time-distance between two objects must be

computable. Every construct, whatever its possible interpretation, is assumed to have a time reference. No assumption was made as to the form of this reference. It can be a subcomponent of that construct directly giving time coordinates, though that need not be the case. The time reference may take a more complex form, e.g. a function whose value depends on the meaning of that construct or its relationship to some other constructs. With respect to this, a function to extract the time reference of any given construct was introduced.

In line with the concept of uniform representation of objects in the Real World, it was tacitly assumed that every object is representable as a process. So long as we describe static properties of such objects, the constructs representing them are called process declarations. According to their role in specific situations they may also be called facts or events to refer mnemonically to their particular meaning. The idea is quite similar to some experiments in λ-calculus to ban free variables (or even variables altogether) by introducing so-called operators. An interesting consequence of this approach is that we can explicitly unify the concept of storing with the concept of processing. Hence, for instance, a construct *'Store number 45 as value for AGE'* would be appropriate rather than *'AGE becomes 45'*.

All actions in the system are represented by process application. The important point is that, in principle, any process may be applied to any construct, including its own representation. This was done in order to incorporate self-modifying systems, i.e. systems whose objects may get changed due to its internal actions. Furthermore, the notion of 'self-regulating' systems has been recognized. The objects of the system are supposed to take necessary actions if and when the conducive circumstances happen. This is a consequence of questioning the concept of 'central control' as being somehow *force majeure*.

In terms of the theory proposed here the 'control' in a system is yet another process application (one or many), possibly designed in the form of complex structure of

processes. These notions constitute, in my view, an attractive enhancement of the von Neumann model of computing.

Furthermore, process applications cooperate with each other passing processed constructs and other information. Their interconnections (also processes) may get changed by the system internally. The concept of Turing machine does not cover the mechanism of application of multiple processes. I needed something else. So instead I specified an abstract machine with two types of channel (each capable of transmitting all sorts of constructs) to handle data and control messages.

Investigation of different forms of precedence and parallelism, their properties and the relationships among them laid down the foundations for a process calculus. A complete set of operations on sentential expressions makes it possible to consider the *actual* temporal relationships occurring among the objects in the Real World. The calculus also provide the means to decompose and integrate process applications. Thus concurrency and other forms of precedence relations can be used for system design. In this sense the theory presented here seems to be a useful extension to the traditional (i.e. restricted to sequential processing) structured techniques.

The question to what extent parallelism and other forms of temporal relationships among processes should affect the data model emerged quite naturally. Investigations carried out within the framework of the relational theory proved that, indeed, the effects of process structure onto the data model were substantial.

Central to the relational model are notions of functional, multivalued and join dependencies. They reflect the semantics of the data and provide a mechanism to transform the initial model to a form free from insert, delete and update anomalies.

However, the provision for non-sequential processing exposed some other anomalies that the above dependencies do not detect. Namely, access anomaly and

store anomaly may occur in otherwise properly normalized relations. Both of these anomalies are undesirable since they may lead to an unnecessary delay in processing or to an uneconomical use of storage.

Analogously to data dependencies, the concept of process dependence offers a way to resolve the above problem. The tangled process dependence occurs when a single relation describes two (or more) entities and independent (though not necessarily concurrent) access is required to both of them. In other words, this happens when a relation contains two (or more) groups of access-independent attributes that are identified by a common primary key. The flat process dependence refers to a situation where a relation is an union of otherwise independently processed sub-relations.

Both the tangled and the flat process dependencies are derivable on semantic grounds by considering the process of abstracting the data, that is creating entity types. The fact that these dependencies can be obtained from either semantic or behavioural considerations underlines the soundness of the initial hypothesis concerning the three aspects of modelling to be 'treated integrally'.

## FURTHER RESEARCH POSSIBILITIES

The theoretical foundations of modelling parallelism (and other forms of temporal relationships) occurring among processes in information systems have, I hope, been investigated sufficiently in this work. Whereas the theory seems to me settled (to some extent at least), there are issues that require further theoretical research. There is also scope for more practical development. Below, the directions for further research are outlined in the following four areas: theoretical research, optimal decomposition, software engineering and computer-aided system development.

- Theoretical research

  As mentioned earlier, the semantic aspects of system description were presented in this work in a simplified form. Therefore a more complete and precise treatment of this subject is desirable. In particular the explicit formulae of semantic functions for common constructs (e.g. process applications such as insert, delete, update, retrieve) need further investigation, so the model of semantics (objects-abstractions-constructs) may be made more precise for a typical information system.

  Another issue worth consideration is semantics and behaviour of recursive process applications. A concept of parallelism of facts may also be investigated, particularly in the context of store anomaly.

- Optimal decomposition

  From Chapter 8 it follows that decomposition of a process application into a number of sub-components interconnected by temporal relationships can be done in many different ways. Therefore the question of whether an optimal decomposition exists and, if so, how can it be done, is of some importance. This problem requires, of course, the optimization criteria to be formulated first.

- Software Engineering

  A purely theoretical approach was taken in this work to analyse and explain the matters concerning parallelism and process modelling for information system description. However, if the results obtained here are to be used in practice of system analysis and design, they will have to be translated into the practitioners' language. That is, in order to become a part of software engineering, the theory will have to be made practicable, in much

the same way as theoretical works on structured programming were turned into practical methods such as JSP.

Second, no method in any kind of engineering is fully appreciated unless it contains a suitable graphical notation capable of recording design decisions and of presenting the results of its actions. Therefore it would be advisable to develop this sort of graphical notation capable of expressing dynamics of system behaviour. Adaptation of Petri Nets [70] seems to be a possible way for achieving this goal.

- **Computer-Aided System Development**

  A description of a life-project in terms of sentential notation may be a complex one - so much so that the system designer will not comprehend it easily. For this reason, an appropriate software package should be used to maintain the knowledge about the project. This software should also be able to detect inconsistencies, omissions and errors that might have occurred in the description. An example of this approach is PSL/PSA [29]. It consists of a general Problem Statement Language (PSL) for describing systems, a database for maintaining the knowledge about the system and a number of procedures (Problem Statement Analyser) to extract this knowledge from the database. The PSL is a general object-property-relationship language that intended to be a tool for describing the characteristics of any system. But it does not support more complicated forms of temporal relationships such as parallelism or mutual exclusiveness. For example it does not allow any 'object' to be viewed both as an entity and as a process, nor allow a relationship to be viewed as an entity. Therefore extensions to PSL/PSA covering these issues would substantially enhance its applicability.

Producing an application package on the basis of a general theory helps in better understanding of how the theory works. Quite often it also causes amendments to be made in theoretical formulae. Most artefacts evolve by gradual improvement: theories sometimes are developed similarly. At some early stage of this present research I attempted to apply the concept of a computer-aided approach to the theory itself. A PROLOG program was written whose task was to check the correctness and consistency of the formulae and corresponding proofs concerning relationships between precedence, parallelism and mutual exclusiveness. Although I do not consider the program itself (a simple procedure for predicate calculus based on the Wang algorithm) as having any particular cognitive value of any academic standing, it led me to certain general conclusions - for instance the distinction between free parallelism and fixed parallelism and the need for a quantified time space.

Undertaking this research I hoped and expected that its results might enhance the theories of system analysis and design and might lead to improved methods and techniques. I believe that, to a modest extent, this goal has been achieved. But having said so I could not resist quoting this passage from G. Spencer Brown's 'Laws of Form' [19]:

> *'... and so on, and so on you will eventually construct the universe, in every detail and potentiality, as you know it now; but then, again, what you will construct will not be all, for by the time you will have reached what now is, the universe will have expanded into a new order to contain what will then be ...'*

## REFERENCES


[1]. **J. R. Abrial**
*Data Semantics*, Database Management, Klimbie & Koffeman (eds), North Holland, IFIP Working Conference on Database Management, 1974

[2]. **Aristotle**
*Methaphysica*, Γ 7, 27, Works, Vol. 8, English translation by W.D. Ross, Oxford 1908

[3]. **T.L. Anderson**
*Modelling Events and Processes at the Conceptual Level*, Proc. ICOD-2, Cambridge, pp. 273-297, 1983

[4]. **T.L. Anderson**
*Modelling Time at the Conceptual Level*, Improving Database Usability and Responsiveness, P. Scheuermann(ed), Academic Press, pp. 151-168, 1982

[5]. **G. R. Andrews**
*Concepts and Notations for Concurrent Programming*, Computing Surveys, Vol.15, No.1, pp. 3-43, 1983

[6]. **V. de Antonelis, A. di Leva**
*DATAID-1: A Database design Methodology*, Information Systems, Vol.10, No.2, pp. 181-195, 1985

[7]. **V. de Antonelis, A. di Leva**
*A Case Study of Database Design Using the Dataid Approach*, Information Systems, Vol.10, No.3, pp. 339-359, 1985

[8]. **C. W. Bachman**
*The Impact of Structured Data Throughout Computer-Based Information Systems*, Information Processing 80, S.H. Lavington(ed), North Holland, pp. 383-394, IFIP 1980

[9]. **J. Backus**
*Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs*, CACM, Vol.21, No.8, pp. 613-641, 1978

[10]. **C. Beeri, P.A. Bernstein, N. Goodmann**
*A Sophisticate's Introduction to Database Normalization Theory*, Proc. of 4th Conf. on Very Large Databases, pp. 113-124, West Berlin, Germany, 1978

[11]. **A. Berztsiss**
*A Conceptual Model Based on Events and Functions*, Syslab Report No.35, May 1985

[12]. **A. Berztsiss**
*The Set-Function Approach to Conceptual Modelling*, Syslab Report No.36, July 1985

[13]. **D. Bjørner, C.B. Jones**
*Formal Specification and Software Development*, Prentice Hall International, 1982

[14]. **A.L. Bossi, C. Ghezzi**
*Using FP as a Query Language for Relational Databases*, Comput. Lang., Vol.9, No.1, pp. 25-37, 1984

[15]. A. **Borgida**, S. Greenspan, J. Mylopoulos
*Knowledge Representation as the Basis for Requirements Specification*,
IEEE Computer, pp.82-90, April 1985

[16]. D. S. **Bowers**
*A Database Architecture for Aggregate-Incomplete Data*,
The Computer Journal, Vol.27, No.4, pp. 294-300, 1984

[17]. E. **Brinksma**
*A Tutorial on LOTOS*, (unpublished)

[18]. S.D. **Brookes**, C.A.R. **Hoare**, A.W. **Roscoe**
*A Theory of Communicating Sequential Processes*,
Journal of the ACM, Vol.31, No.3, pp. 560-590, 1983

[19]. G.S. **Brown**
*Laws of Form*, Pitman, 1969

[20]. J. A. **Bubenko** (Jr)
*Information Modelling in the Context of System Development*,
Information Processing 80, S.H. Lavington(ed), North Holland, IFIP 1980

[21]. E. **Charniak**, D. **McDermott**
*Introduction to Artificial Intelligence*, Addison-Wesley, 1985

[22]. P.P.S. **Chen**
*The Entity-Relationship Model - Toward a Unified View of Data*,
ACM TODS, Vol.1, No.1, 1976

[23]. A. **Church**
*The Calculi of lambda-conversion*, Princeton University Press,
Princeton, 1941

[24]. E.F. **Codd**
*A Relational Model of Data for Large Shared Data Banks*,
CACM, Vol.13, No.6, pp. 377-387, 1970

[25]. E. F. **Codd**
*Extending the Database Relational Model to Capture More Meaning*,
CACM, Vol.4, No.4, pp. 397-434, 1979

[26]. C.J. **Date**
*An Introduction to Database Systems*,
Vol. I and II, Addison-Wesley, IV edition, 1986

[27]. C. **Delobel**
*An Overview of the Relational Data Theory*, Information Processing 80,
S.H. Lavington(ed), pp. 413-425, North Holland, IFIP 1980

[28]. E.W. **Dijkstra**
*Cooperating Sequential Processes*,
Programming Languages, NATO Advanced Study Institute, F. Genuys(ed),
Academic Press, pp. 43-112, 1968

[29]. **DEC-T810**: *Information System Development*,
PSL/PSA User Course, Highfield Park, September, 1984

[30]. A.L. **Furtado**, T.S.E. **Maibaum**
*An Informal Approach to Formal (Algebraic) Specifications*,
The Computer Journal, Vol.28, No.1, pp. 59-67, 1985

[31]. G. Gardarin, E. Gelenbe
*New Applications of Data Bases*, Academic Press, 1984

[32]. M.A. Gray
*Implementing Unknown and Imprecise Values in Databases*,
Proc. of BNCOD-1, pp. 146-158, Pentech Press, Cambridge, July, 1981

[33]. J.J. van Griethuysen
*Concepts and Terminology for the Conceptual Schema and the Information Base*,
ISO TC97/SC5/WG3 - N695, 1982

[34]. R.Haux, U. Eckert
*Nondeterministic Dependencies in Relations: An Extension of the Concept of Functional Dependency*, Information Systems, Vol.10, No.2, pp. 139-148, 1985

[35]. C.A.R. Hoare
*Communicating Sequential Processes*, CACM, Vol.21, No.8, pp. 666-677, 1978

[36]. C.A.R. Hoare
*A Calculus of Total Correctness for Communicating Processes*,
Oxford University Computing Laboratory, PRG-23, April 1981

[37]. C.A.R. Hoare
*A Model for Communicating Sequential Processes*,
Oxford University Computing Laboratory, PRG-22, June 1981

[38]. C.A.R. Hoare
*Notes on Communicating Sequential Processes*,
Oxford University Computing Laboratory, PRG-33, August 1983

[39]. C.A.R. Hoare
*Communicating Sequential Processes*, Prentice Hall International, 1985

[40]. D.R. Hofstadter
*Gødel, Escher, Bach: An Eternal Golden Braid*, The Harvester Press Ltd., 1979

[41]. R. Hull, C.K. Yad
*The Format Model: A Theory of Database Organization*,
Journal of the ACM, Vol.31, No.3, pp. 518-537, 1984

[42]. W. Kent
*Data and Reality*, North Holland, 1978

[43]. W. Kent
*Consequences of Assuming a Universal Relation*,
ACM TODS, Vol.6, No.4, pp. 539-556, 1981

[44]. S. Khosla, T.S.E. Maibaum, M. Sadler
*Database Specification*,
Proc. of IFIP TC2 Working Conference on Database Semantics,
Hasselt, Belgium,January 1985

[45]. R. King, D. McLeod
*The Event Database Specification Model*, Improving Database Usability and Responsiveness, pp. 299-320, P. Scheuermann(ed), Academic Press, 1982

[46]. A. Kobsa
*Knowledge Representation: A Survey of Its Mechanisms; A Sketch of Its Semantics*, Cybernetics and Systems, Vol.15, Nos.1-2, pp. 41-89, 1984

[47]. K. Kuratowski, A. Mostowski
*Set Theory*, PWN, Warsaw, 1968

[48]. L. Lamport
*The Mutual Exclusion Problem - Part I: A theory of Interprocess Communication*,
Journal of the ACM, Vol.33, No.2, pp. 313-326, 1986

[49]. L. Lamport
*The Mutual Exclusion Problem - Part II: Statement and Solution*,
Journal of the ACM, Vol.33, No.2, pp. 327-348, 1986

[50]. U.W. Lipeck, H-D. Ehrich, M. Gogolla
*Specifying Admissibility of Dynamic Database Behaviour Using Temporal Logic*,
IFIP WG8.1 Conf. on Inf. Systems: Theoretical and Formal Aspects, pp.145-157,
North Holland, 1985

[51]. T. Imielinski, W.Lipski (Jr)
*Incomplete Information in Relational Databases*,
Journal of the ACM, Vol.31, No.4, pp. 761-791, 1984

[52]. B.E. Jacobs
*On Database Logic, Journal of the ACM*, Vol.29, No.2, pp. 310-322, 1982

[53]. R. Johnson
*Integrating Data and Metadata to Create a Smart Database*,
Thames Polytechnic (un-published)

[54]. R.N. Maddison (ed)
*Information Systems Methodologies*, BCS Monographs in Informatics,
Wiley Heyden, 1983

[55]. R.N. Maddison (ed)
*Information System Development: A Flexible Framework*,
BCS, ISAD WP Journal of Development, 1984

[56]. R.N. Maddison (ed)
*Information Systems Methodologies*, Inf. State-of-the-Art Report, Vol.13, 3, 1985

[57]. T.S.E. Maibaum
*Database Instances, Abstract Data Types and Database Specification*,
The Computer Journal, Vol.28, No2, pp. 154-161, 1985

[58]. D. Maier, J.D. Ullmann, M.Y. Vardi
*On the Foundations of the Universal Relation Model*,
ACM TODS, Vol.9, No.2, pp. 283-308, 1984

[59]. M.G. Main, D.B. Benson
*Denotational Semantics for "Natural" Language Question-Answering Programs*,
American Journal of Computational Linguistics, Vol.9, No.1, pp. 11-21, 1983

[60]. N.J. Martin
*The Construction of Interfaces to Triple Based Databases*,
Research Report NJM/01/84, Birkbeck College, University of London, 1984

[61]. W.J. Milne
*A Framework for the Investigation of a Spatial Database*,
The Computer Journal, Vol.24., No.1, pp. 52-55, 1981

[62]. C. Morris
*Signs, Language and Behaviour*, G. Brazillier, New York, 1946

[63]. G.M. Nijssen
*From databases towards knowledge bases*, pp. 115 - 131,

[64]. G.M. Nijssen
*On Experience with Large Scale Teaching and Use of Fact-Based Conceptual Schemas in Industry and University*,
Proc. of IFIP TC2 Working Conference on Database Semantics,
Hasselt, Belgium,January 1985

[65]. T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (eds)
*Information Systems Design Methodologies: A Comparative Review*,
Proc. of the IFIP TC.8 Conference (CRIS), North Holland, 1982

[66]. T.W. Olle, J.G. Sol, C.J. Tully (eds)
*Information Systems Design Methodologies: A Feature Analysis*,
Proc. of the IFIP WG 8.1 Conference (CRIS 2), North Holland, 1983

[67]. L. Orman
*Design Criteria for Functional Databases*,
Information Systems, Vol.10, No.2, pp. 207-217, 1985

[68]. L. Orman
*A Familial Specification Language for Database Application System*,
Computer Languages, Vol.8, No.3/4, 1983

[69]. OU- M352 : *Computer Based Information Systems*,
The Open University Press, 1980

[70]. J.L. Peterson
*Petri Nets*, Computing Surveys, Vol.9, No.3, pp. 223-252, 1977

[71]. A.Ramon
*Information Derivability Analysis in Logical Information Systems*,
CACM, Vol.26, No.11, pp. 933-938, 1983

[72]. G. Richter
*Clocks and their Use for Time Modelling*,
IFIP WG8.1 Conf. on Inf. Systems: Theoretical and Formal Aspects,
pp. 49-66, North Holland, 1985

[73]. G.D. Ritchie, F.K. Hanna
*Semantic Networks - A General Definition and a Survey*,
Information Technology: Research and Development (1983), 2, pp. 187-231

[74]. H. Robinson, M. Newton
*The Capture of Meaning and the Relational Data Model*,
Technical Report 85/13, Computing Discipline, The Open University

[75]. N. Roussopoulos, R.T. Yeh
*An Adaptable Methodology for Database Design*, IEEE Computer, May 1984

[76]. U. Schiel
*Time Dimension in Information Systems*,
IFIP WG8.1 Conf. on Inf. Systems: Theoretical and Formal Aspects,
pp. 67-76, North Holland, 1985

[77]. H.A. Schmid, J.R. Svenson
*On the Semantics of the Relational Data Model*,
Proc. ACM SIGMOD Conf. on Management of Data, pp.211-223,
San Jose, California, May 1975

[78]. M.E. Senko
*Information Systems: Records, Relations, Sets, Entities and Things*,
Information Systems, Vol.1, pp. 3-13, 1975

[79]. A. Sernadas
*Logical Procedure Definition for Information System Specifications*,
PhD Thesis, University of London (LSE), 1980

[80]. A. Sernadas
*SYSTEMATICS: Its Syntax and Semantics as a Query Language (1)*,
The Computer Journal, Vol.24, No.1, 1981

[81]. A. Sernadas
*SYSTEMATICS: Its Syntax and Semantics as a Query Language (2)*,
The Computer Journal, Vol.24, No.2, 1981

[82]. C. Sernadas, A. Sernadas
*Conceptual Modelling Abstraction Mechanisms as Parametrised Theories in Institutions*, INFOLOG PR20, August 1984

[83]. M.J.R. Shave
*Entities, Functions and Binary Relations: Steps to a Conceptual Schema*,
The Computer Journal, Vol.24, No.1, pp. 42-46, 1981

[84]. A. Solvberg, C.H. Kung
*On Structural and Behavioural Modelling of Reality*,
Proc. of IFIP TC2 Working Conference on Database Semantics,
Hasselt, Belgium,January 1985

[85]. R.K. Stamper
*Towards a Semantic Normal Form*, Database Architecture,
Bracci & Nijssen (eds), pp. 317-339, North Holland, SRC/SSRC 1979

[86]. S.K. Stanczyk
*A Relational Database Project for the Motorway Information System*,
MSc Thesis, University College London, 1980

[87]. S.K. Stanczyk, R.N. Maddison
*Modelling Parallel Processes for Databases*,
Technical Report 87/2, Computing Discipline, The Open University

[88]. J. Stoy
*The Scott-Strachey Approach to Programming Language Theory*,
MIT Press, Cambridge, Massachussetts, 1977

[89]. B. Sundgren
*Conceptual Foundations of the Infological Approach to Databases*,
Database Management, Klimbie & Koffeman(eds), North Holland,
IFIP Working Conference on Database Management, 1974

[90]. P.S.G. Swinson, F.C.N. Pereira, A. Bijl
*A Fact Dependency System for the Logic Programmer*,
Computer Aided Design, Vol.15, No.4, pp. 235-243, 1983

[91]. A. Tarski
*Logic, Semantics, Methamathematics*,
Oxford University Press, 1956

[92]. J.D. Ullmann
*Principles of Database Systems*, Pitman, 1983

[93]. **M. Vetter,** R.N. Maddison
*Database Design Methodology*, Prentice Hall International, 1981

[94]. **A. Waldron**
*Principles of Language and Mind*, Routledge & Kegan Paul, 1985

[95]. **H.K.T. Wong, J. Mylopoulos**
*Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management*, Informatics, Vol.15, No.3, pp. 344-383, 1977

# TABLE OF SYMBOLS

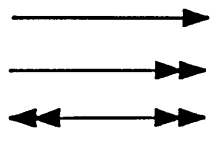
## PREDICATE CALCULUS

| | |
|---|---|
| **there is** | existential quantifier |
| **for all** | universal quantifier |
| **not** | negation |
| **and** | conjunction |
| **or** | disjunction |
| **implies** | implication |
| **iff** | equivalence |

## SETS AND FUNCTIONS

| | |
|---|---|
| $\in$ | set membership |
| $\cup$ | union |
| $\cap$ | intersection |
| $\subset$ | inclusion |
| $\subseteq$ | inclusion or equality |
| $\sim$ | complement |
| $\{\}$ | empty set |
| $\times$ | Cartesian product |
| : | such that |
| $\rightarrow$ | maps into |
| $\leq_t$ | linear ordering in time space |
| $<\ldots, t^i_k, t^i_{k+1}, t^i_{k+2}, \ldots>$ | sequence such that $\ldots \leq_t t^i_k \leq_t t^i_{k+1} \leq_t t^i_{k+2} \leq_t \ldots$ |

## RELATIONSHIP TYPES

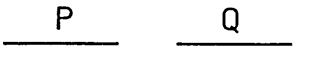
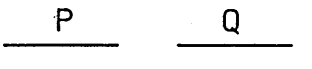
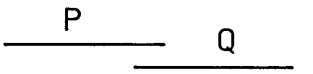
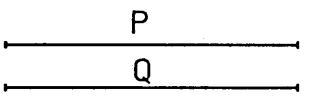
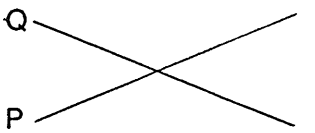
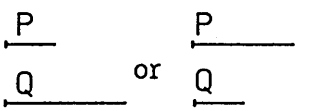
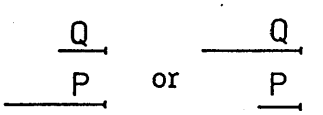
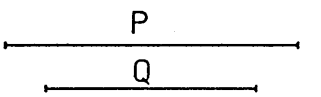


one-to-one

one-to-many

many-to-many

## PROCESS CALCULUS

| SYMBOL | SYNTACTIC EQUIVALENT | GRAPHICAL EQUIVALENT | READS AS | EXAMPLE OF USE |
|---|---|---|---|---|
| >- | **prec** |   | precedes | P >- Q, P prec Q |
| -< | **fllw** |   | follows | Q -< P, Q fllw P |
| >:- | **sync** |   | overlaps | P >:-Q, P sync Q |
| ‖ | **prls** |   | strictly parallel | P ‖ Q, P prls Q |
| ∫∫ | **prrl** | - | freely parallel | P ∫∫ Q, P prrl Q |
| ⊥ | **clsh** |   | clashes | P ⊥ Q, P clsh Q |
| ! : | **strt** |   | start together | P ! : Q, P strt Q |
| : ! | **endt** |   | end together | P : ! Q, P endt Q |
| >< | **witn** |   | within | Q >< P, Q witn P |

# APPENDIX

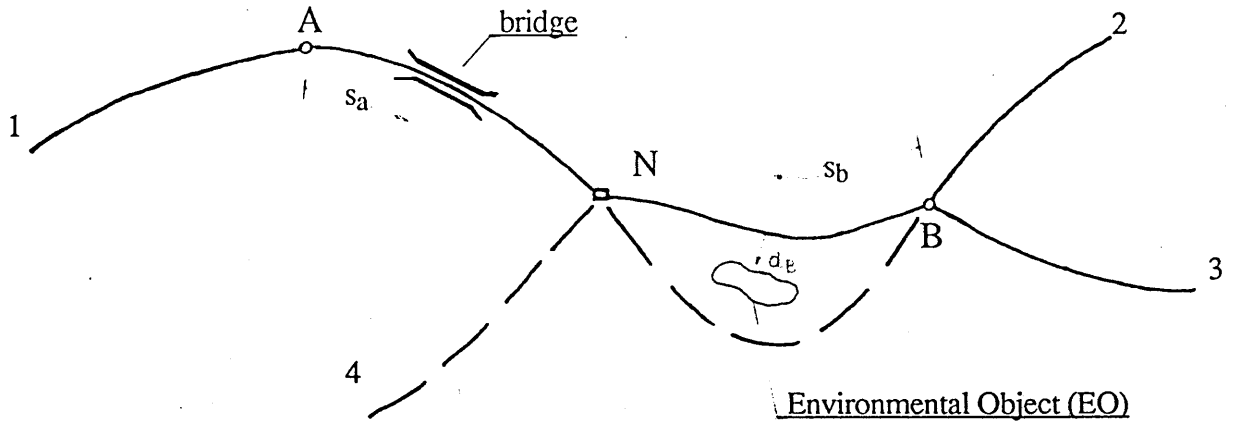
The aim of this appendix is to demonstrate how some major concepts and notational devices developed in this thesis can be applied to describe a real system. As an object of this demonstration I have chosen a limited part of an existing Road Database (RDB) containing information on some 100,000 km of road network.

For the sake of clarity only a very small part of RDB will be considered - a complete description would have unnecessarily complicated the matter. In any case such description is a serious task on its own and considerably exceeds the scope of this appendix. Nonetheless, some details on the system as a whole are presented below to facilitate the understanding of problems considered.

Conceptually, the RDB contains three classes of information:

- the topology of the road network
- characteristics of road segments
- characteristics of various environmental objects related somehow to the road segments

The general layout of data stored in the RDB is presented in Fig. A1. The road network is modelled by a graph whose nodes represent certain defined points (such as junctions), physically existing on the network. These nodes, in fact, may themselves be complex objects (e.g. multilevel junctions) but this aspect is omitted here. The arcs of the graph represent road segments, i.e. road stretches connecting two nodes.




**NODES**

| Code | Time-space coordinates |
|---|---|
| A | $(X_A, Y_A, Z_A, t_A)$ |
| B | $(X_B, Y_B, Z_B, t_A)$ |
| | . |
| | . |
| | . |
| N | $(X_N, Y_N, Z_N, t_N)$ |

**ARCS**

| Node1 | Node2 | Time-coordinate |
|---|---|---|
| A | 1 | $t_{A\text{-}1}$ |
| A | B | $t_{A\text{-}B}$ |
| B | 2 | $t_{B\text{-}2}$ |
| B | 3 | $t_{B\text{-}3}$ |
| | . | |
| A | N | $t_{A\text{-}N}$ |
| N | 4 | $t_{N\text{-}4}$ |
| N | B | $t_{N\text{-}B}$ |

**ROAD CHARACTERISTICS**

| Segment | Technical details | Time-coordinate |
|---|---|---|
| A-1 | $details_{A\text{-}1}$ | $t(details_{A\text{-}1})$ |
| A-B | $details_{A\text{-}B}$ | $t(details_{A\text{-}B})$ |
| B-2 | $details_{B\text{-}2}$ | $t(details_{B\text{-}2})$ |
| B-3 | $details_{B\text{-}3}$ | $t(details_{B\text{-}3})$ |
| | . | |
| | . | |
| A-N | $details_{A\text{-}N}$ | $t(details_{A\text{-}N})$ |
| N-4 | $details_{N\text{-}4}$ | $t(details_{N\text{-}4})$ |
| N-B | $details_{N\text{-}B}$ | $t(details_{N\text{-}B})$ |

**ENVIRONMENTAL OBJECTS**

| Object | Location | Technical Details | Time-coordinate |
|---|---|---|---|
| Bridge | $s_A$ | details | $t(s_A)$ |
| EO | $(s_B, d_B)$ | details | $t(s_B, d_B)$ |

Fig. A1

This system of arcs and nodes has been designed for the sake of unambiguous identification of any point, stretch or area being of interest to any information system concerning roads and traffic, irrespectively of whether or not that system uses at present the information stored in the database.

Each road segment is described by a (large) number of attributes representing horizontal and vertical alignment, technical and structural details, and traffic flow and traffic accidents.

Environmental objects include bridges, viaducts, ferry berths and other, more complex entities such as service areas, telecommunication lines and electricity lines. The important fact is that although some of these objects are not maintained by the road administration they, nevertheless, have a considerable impact on the traffic space and capacity of the roads.

Virtually all data items are - to a lesser or greater extent - dependent on time, with a different degree of volatility. For instance, the traffic conditions on a particular road segment may change within hours while structural parameters remain stable over a longer period of time. As a general rule, however, no information held in the database should ever be destroyed, even if the corresponding object in the Real World ceases to exist.

The system serves as a sole source of information to three distinctive categories of users:

- specific sub-systems based on established processing algorithms (eg traffic analysis, shortest path routes)
- various kinds of professionals (engineering, management, researchers, planners) implementing new or ad-hoc algorithms
- casual users such as politicians, authorities and general public

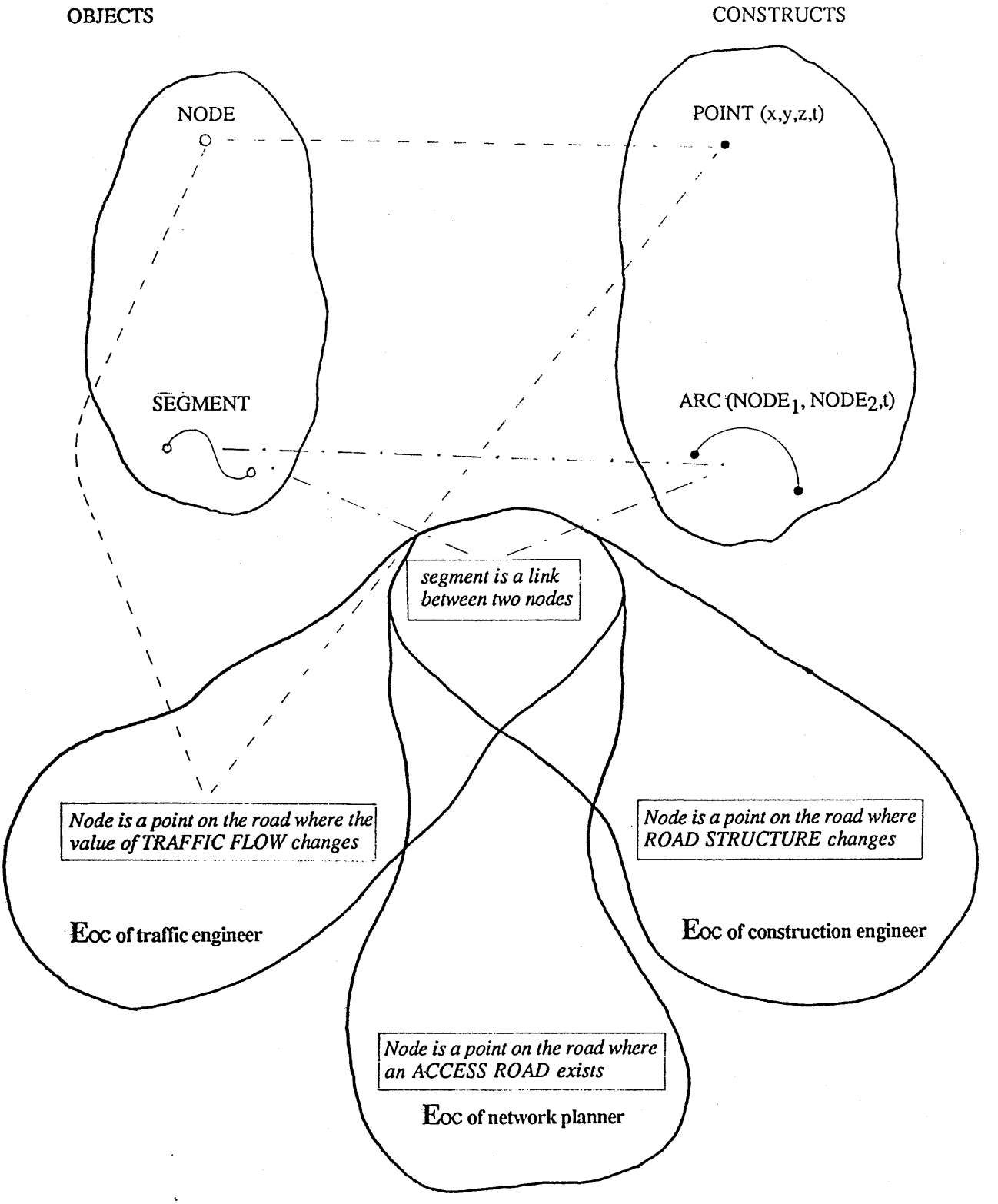

OBJECTS
CONSTRUCTS
NODE
POINT (x,y,z,t)
SEGMENT
ARC (NODE1, NODE2,t)
segment is a link between two nodes
Node is a point on the road where the value of TRAFFIC FLOW changes
Eoc of traffic engineer
Node is a point on the road where ROAD STRUCTURE changes
Eoc of construction engineer
Node is a point on the road where an ACCESS ROAD exists
Eoc of network planner

Fig. A2

Apart from typical (and therefore fairly simple) database interrogations two other kinds of user-oriented processing are of particular importance. The first one is a so-called complex transformation. It occurs most frequently whenever a user (typically a sub-system) requires some sort of an abstraction of the total content of the database. Examples are: producing a road sub-network that fulfils certain criteria and drawing maps in a scale different from that which is a base for network topology. The second kind of processing includes combinatorial retrievals. These are characterised by little input/output but involving sometimes the whole database to be accessed.

However, the most important activity (and, indeed, the primary objective of the system) is the maintenance of data. Data collection in this case is extremely expensive and very often involves sophisticated equipment. Updating routines, in turn, are invariably complex and resource consuming. Thus, they need to be considered with a great care and designed very efficiently. A particular updating is therefore described in this appendix to illustrate the usefulness of some findings of this thesis.

In the RDB one global time and a number of local time systems exist. Quite naturally, the universal time (described by the Gregorian time scale) has been selected as the system time ($T^o$). The local time systems include:

- system updating time T established at the beginning of system operation
  resolution ($T^o$) = resolution (T)
- topology updating time with resolution 3 months
- complex transformation time, resolution 6 months
- combinatorial retrievals time, resolution 1 day
- source-destination traffic measurements, resolution 1 hour
- automatic traffic survey, resolution 1 second

The data model for the RDB requires the semantics of its elements to be established. Fig.A2 presents an attempt to define the meaning of two basic entities - NODE and SEGMENT within the framework of the semantic system, suggested in Chapter 4. The

structure depicted in Fig A2 was meant to expose the fact that the Environments of Comprehension of three different groups of users overlap only partially having different descriptions for one of the basic concepts of the RDB - the concept of node. All three groups agree though in description of a road segment.

Let's consider now a particular kind of an updating routine that necessitates changes in all constructs depicted in Fig.A1. Suppose initially there was just AB; then a new node N together with two connections N-4 and N-B was built; and at the same time the 'old' connection N-B ceased to exist. The RDB needs to be updated and the following processes must be executed:

$P_1$: insert 'new' node N to **NODES**

$P_2$: change time-coordinate for the 'old' arc AB in **ARCS**

$P_3$: insert 'new' arcs AN, NB, N4 to **ARCS**

$P_4$: change time coordinates for details of AB in **ROAD CHARACTERISTICS**

$P_5$: insert details on AN, NB, N4 to **ROAD CHARACTERISTICS**

$P_6$: modify information about the Environmental Objects

Since the RDB is continually operational, an independent process Q requiring access to information on AB may be expected at any time.

The processes $P_1$, $P_2$ and $P_4$ are atomic, but $P_3$, $P_5$ and $P_6$ can be decomposed as follows. The process $P_3$ is a collection of the following sub-processes:

$$P_3 = P_{3.AN} \iint P_{3.NB} \iint P_{3.N4}$$

where $P_{3.XY}$ means '*insert tuple representing segment* XY *into* ARCS'.

Similarly, the process $P_5$ consists of:

$$P_5 = P_{5.AN} \iint P_{5.NB} \iint P_{5.N4}$$

where $P_{5.XY}$ means '*insert details on segment* XY *into* ROAD CHARACTERISTICS'.

Finally, the process of $P_6$ is composed of:

$$P_6 = (P_{6.COPY} \parallel P_{6.NEW\text{-}TIME}) >\!- P_{6.NEW\text{-}LOC} >\!- P_{6.MOD\text{-}TIME}$$

where

$P_{6.COPY}$ = retain those details about EO except the details to be changed

$P_{6.NEW\text{-}TIME}$ = create new time coordinates for the changed part of the EO-duplicate

$P_{6.NEW\text{-}LOC}$ = modify the above duplicate to refer the EO to the 'new' segment NB

$P_{6.MOD\text{-}TIME}$ = modify time reference in EO-original to mark it as now being a 'historical' value

Typically the sequence of processes is: $(P_1 >\!- P_3 >\!- P_5 >\!- P_6 >\!- P_4 >\!- P_2) \perp Q$ which after substitutions yields the following expression to analyse:

$(P_1 >\!- (P_{3.AN} \iint P_{3.NB} \iint P_{3.N4}) >\!- (P_{5.AN} \iint P_{5.NB} \iint P_{5.N4}) >\!- (P_{6.COPY} \parallel P_{6.NEW\text{-}TIME}) >\!- P_{6.NEW\text{-}LOC} >\!- P_{6.MOD\text{-}TIME}) >\!- P_4 >\!- P_2) \perp Q$

For instance, some possible decompositions of Q may be worth considering since the following conditions are reasonable: $P_{3,N4} \iint Q$, $P_{5.N4} \iint Q$ and $P_6 \iint Q$.

On current computer systems using sequential processes in a conventional programming languages the strict parallelism, such as this between the parts of $P_6$, cannot be implemented. In practice one might do $P_{6.COPY}$ by duplicating the complete details about EO, and then adjusting the copy. Thus after copying, one does $P_{6.NEW\text{-}TIME}$ by assignment statements which ovewrite the now-obsolete details,

while retaining the unchanged details in the copy. To avoid the clash with Q some form of record locking is required.